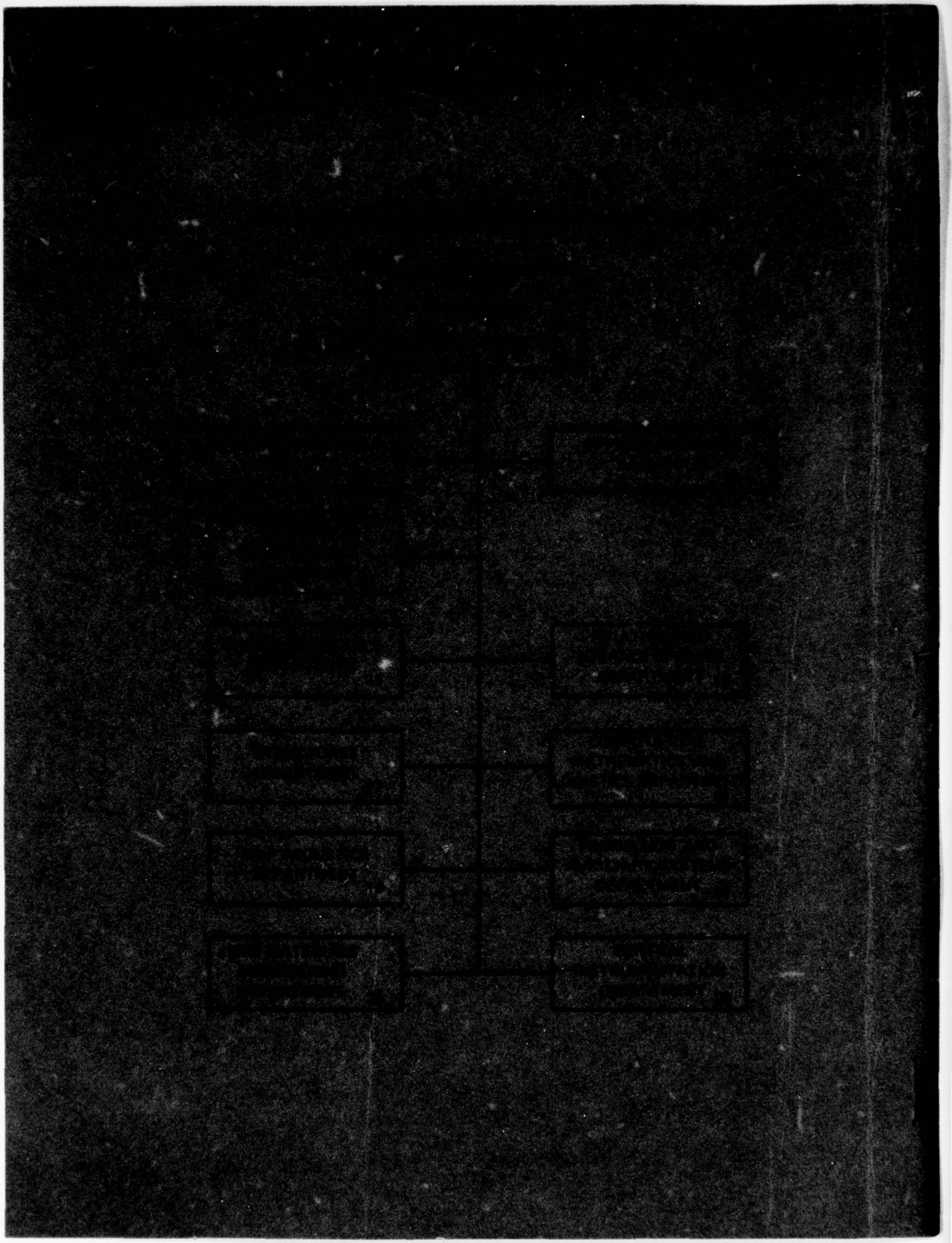
AD-A055 097 DAVID W TAYLOR NAVAL SHIP RESEARCH AND DEVELOPMENT CE--ETC F/G 9/2
FEASIBILITY STUDY FOR INCORPORATING A DATA STRUCTURE DEFINITION--ETC(U)
MAY 78 I S ZARITSKY

UNCLASSIFIED DTNSRDC-78/045 NL

1 OF 2
AD
A055097



UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE

READ INSTRUCTIONS BEFORE COMPLETING FORM

1. REPORT NUMBER: DTNSRDC-78/045

2. GOVT ACCESSION NO.

3. RECIPIENT'S CATALOG NUMBER

4. TITLE (and Subtitle): FEASIBILITY STUDY FOR INCORPORATING A DATA STRUCTURE DEFINITION AND MANIPULATION FACILITY WITHIN THE COMRADE DATA MANAGEMENT SYSTEM

5. TYPE OF REPORT & PERIOD COVERED: Final rept.

6. PERFORMING ORG. REPORT NUMBER

7. AUTHOR(s): Irving S. Zaritsky

8. CONTRACT OR GRANT NUMBER(s): F53532, ZF53532001

9. PERFORMING ORGANIZATION NAME AND ADDRESS: David W. Taylor Naval Ship Research and Development Center, Bethesda, Maryland 20084

10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS: (See reverse side)

11. CONTROLLING OFFICE NAME AND ADDRESS

12. REPORT DATE: May 1978

13. NUMBER OF PAGES: 116 P.

14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office)

15. SECURITY CLASS. (of this report): UNCLASSIFIED

15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

APPROVED FOR PUBLIC RELEASE: DISTRIBUTION UNLIMITED

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

COMRADE
Computer-Aided Design
Data Management
Data Base Management System
Information Retrieval System
Graph
Data Base
Associative Memory
Data Definition Language

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

A scheme is described for enhancing the COMRADE (Computer-Aided Design Environment) Data Management System. This scheme would produce benefits in data management efficiency, user convenience, power, and cost effectiveness by representing the data structure of a COMRADE data base apart from the data records and by adding a system specifically designed to handle pointer information. In particular, such techniques would:

(Continued on reverse side)

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE
S/N 0102-LF-014-6601

UNCLASSIFIED
SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

387682

(Block 10)

62760N

Program Element 6276N
Project F53532
Task Area ZF53532001
Work Unit 1-1808-009

(Block 20 continued)

- reduce COMRADE use of disk I/O for data block relationships;
- simplify the organization and administration of the data base;
- enable the use of a powerful data-definition/data manipulation language;
- enable the use of an inferential search mechanism; and
- permit existing programs involving pointer relationships to remain essentially unchanged.

The degree of effectiveness achieved under this scheme can be further enhanced by giving the data base administrator a greater role in developing and maintaining the data base in a way to capitalize on the greater flexibility provided.

The procedures involved in implementing the proposed scheme and the benefits to be realized from such a scheme are illustrated by describing a hypothetical COMRADE/GIRS system. GIRS (Graph Information Retrieval System) is an in-house developed system written in FORTRAN that is particularly efficient at manipulating pointers. It is already operable on the CDC 6700, the PDP-11/45, and the UNIVAC 1108, and is easily portable to other machines.

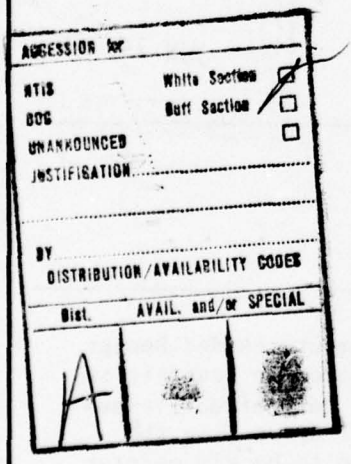


TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ABSTRACT

A scheme is described for enhancing the COMRADE (Computer-Aided Design Environment) Data Management System. This scheme would produce benefits in data management efficiency, user convenience, power, and cost effectiveness by representing the data structure of a COMRADE data base apart from the data records and by adding a system specifically designed to handle pointer information. In particular, such techniques would

- reduce COMRADE use of disk I/O for data block relationships,
- simplify the organization and administration of the data base,
- enable the use of a powerful data-definition/ data manipulation language,
- enable the use of an inferential search mechanism, and
- permit existing programs involving pointer relationships to remain essentially unchanged.

The degree of effectiveness achieved under this scheme can be further enhanced by giving the data base administrator a greater role in developing and maintaining the data base in a way to capitalize on the greater flexibility provided.

The procedures involved in implementing the proposed scheme and the benefits to be realized from such a scheme are illustrated by describing a hypothetical COMRADE/GIRS system. GIRS (Graph Information Retrieval System) is an in-house developed system written in FORTRAN that is particularly efficient at manipulating pointers. It is already operable on the CDC 6700, the PDP-11/45, and the UNIVAC 1108, and is easily portable to other machines.

INTRODUCTION

The COMRADE (Computer-Aided Design Environment) software developed at the David W. Taylor Naval Ship Research and Development Center in support of the CASDAC (Computer-Aided Ship Design and Construction) effort being conducted by NAVSEA (Naval Sea Systems Command) consists of three major components--the COMRADE Executive, the COMRADE Data Management System, and the COMRADE Design Administration System.[1] This report describes a method for enhancing data managment efficiency, user convenience, power, and cost effectiveness of the data management component.

The concept is simple. Under the present COMRADE arrangement, much of the information contained in a particular data block of a common data base is irrelevant for a specific task, since a data base must serve different users having different requirements. B. Thomson[2] speaks of the common data base used for ship design in the following way:

---

[1] Gorham, W. and T. Rhodes, "COMRADE, Computer Aided Design Environment Project, An Introduction, DTNSRDC Report 76-0001 (Nov 1976). A complete listing of references is given on page 111.

[2] Thomson, B., "Plex Data Structure for Integrated Ship Design," Presented at the 1973 National Computer Conference, American Federation of Information Processing Societies, New York, pp. 347-352 of the Proceedings (Jun 1973).

> "The SDF (Ship Design File) solves the data communications problem by providing a single common, current compendium of design information... The various design disciplines view the ship from different perspectives and require common data to be organized in a variety of different ways. The SDF employs a plex structure* which relies upon sundry types of pointers to connect data blocks in the various relationships required by the user engineers and the applications programs."

The necessity for providing different pointer information for each user results in the data base becoming large and cumbersome to search. Moreover, once the structures (Block Type Definitions) which define the contents of the data blocks have been determined and used, they are extremely difficult to modify. Under the current COMRADE scheme, pointer searches and updates involving pointer searches consume a large amount of disk I/O in traversing non-hit pointers, since each of the data blocks containing relevant pointers must be brought into main memory even though most of the contents are unrelated to that search. Such a necessity makes it impractical for the user to either recognize or report on a large file structure. Moreover, pointer information takes up space better reserved for data.

A scheme is available which can nullify these problems. Under this scheme, all pointer information is extracted from the data blocks and collected into a common area which can exist either as part of the data file or as a separate file

---

* A plex structure is the most general form of data structure in which any given node may be related to any other.

established exclusively for pointers. This collection of pointers can be easily traversed or manipulated by a system especially designed for that purpose.

A data file could hold several different descriptions of the data structure (several different pointer collections), including, if desired, a complete description of the data base. This requirement of multiple views of the data base is discussed in Jefferson and Bandurski.[3]

> "Different users desire different information from the database, are familiar with different naming conventions and levels of detail, are permitted to read and alter different parts of the database, and impose different types of consistency constraints upon the database. It is convenient, then, to have different views, or descriptions, of the same database for different users. Further, a program may remain stable as the database structure changes, if the program is provided with an unchanging view of the database.
>
> The user's view of the database may be simplified by eliminating unnecessary relations, records, and fields--essentially, by providing him with a subset of the database--and possibly by renaming the remaining relations and fields."

Under the approach proposed within this report*, the data records would be maintained separately from the pointer-information relating to the data records. The data records

---

* A network model, not the relational model of Bandurski and Jefferson. See their paper[4] for the advantages and disadvantages of the relational model.

[3] Bandurski, A. and D. Jefferson, "Enhancements to the Relational Model for Computer Aided Ship Design," DTNSRDC Report 4759 (Oct 1975).

[4] Bandurski, A. and D. Jefferson, "Data Description for Computer-Aided Ship Design," DTNSRDC Report 4759 (Sept 1975).

would be compressed and the excluded pointer data collected into a concise representation of the file structure so that a system designed for graph processing could operate on several pointer relationships without accessing the data blocks.

Thomson cites the need for a system which will allow flexibility in referencing. In his description of the SDF he has this to say:

> "The most significant characteristic of the SDF is that most data elements have several distinct relationships to other data elements. For instance, a piece of electronic equipment may belong to a sonar system, may be located in the Sonar Control Room, may be classified in weight group 412, and may be physically connected to various other components in the sonar system and in the electrical distribution system, the water cooling system, and the fire control system. The data structure must allow the electronic component to be referenced via any of the above relationships. This requirement reflects the inclination of various disciplines to view the ship from different perspectives, and it dictates a high degree of interconnectivity and flexibility within the SDF."

Under the proposed system, the cost of maintaining the data records would continue to be shared by all users, but the cost of maintaining a particular Data Definition Dictionary (DDD) defined by the Data Base Administrator (DBA) would be borne by the particular individual for whom it was developed, thus resulting in a more equitable distribution of the cost.

A COMRADE data base as presently structured is shown in Figure 1.

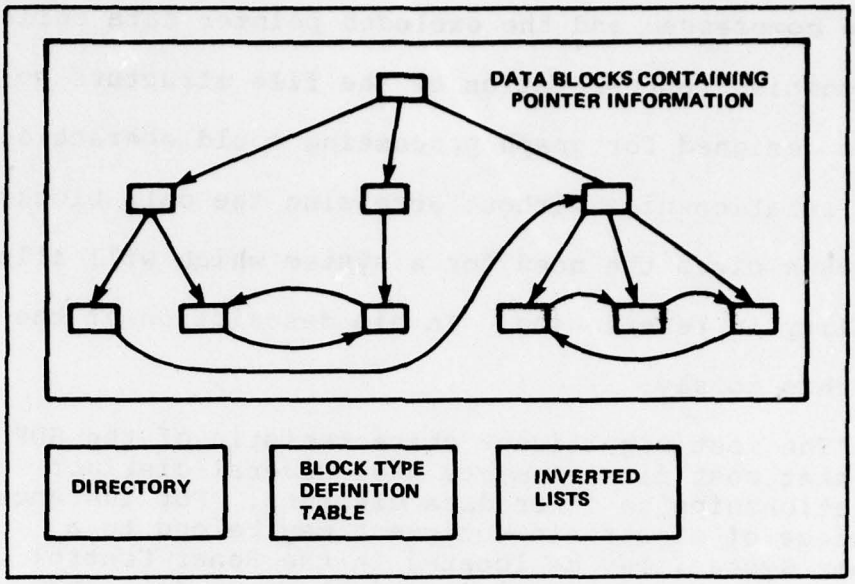


Figure 1 - Current Structure of a COMRADE Data Base

Under the proposed system, the structure of this same data base would be changed as shown in Figure 2.

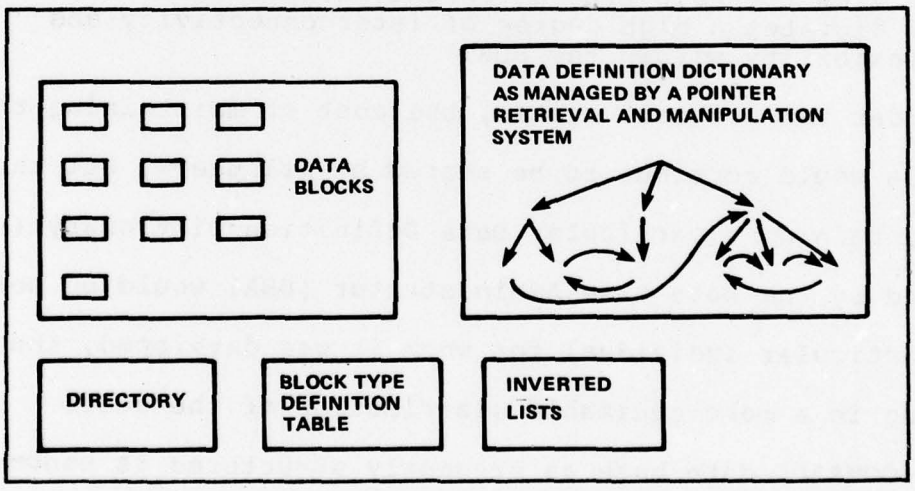


Figure 2 - Proposed Structure of a COMRADE Data Base

Note that the only component affected by this restructuring would be pointer structure and its attendant subroutines.

To provide maximum flexibility for the manipulation of pointers in an arbitrarily directed (plex) graph, an efficient system for storing, retrieving, and manipulating information in main memory is needed. Thomson has this to say about the data structure design needs of ISDS (Integrated Ship Design System), a prime user of COMRADE:

> "Early experience with a list structured Ship Design File revealed that such a structure was too restrictive for the requirements of ISDS...It was accepted that a very general plex structure was required."

Several systems exist which might be applicable for use within COMRADE for pointer manipulation (see later section entitled "Satisfying an Imprecise Query"); however, the paged version of the in-house developed system known as GIRS (Graph Information Retrieval System[5] [6] is ideal for representing and processing "plex", or arbitrary, graph structures. Moreover, GIRS can be adapted to different computing systems quite easily, and is currently available on the CDC 6700, the PDP-11/45, and the Univac 1108. For these reasons, the proposed scheme is described in terms of a hypothetical COMRADE/GIRS configuration.

GIRS used in conjunction with COMRADE offers a number of advantages:

---

[5] Berkowitz, S., "Design Trade-Offs for a Software Associative Memory," DTNSRDC Report 3531 (May 1973).

[6] Zaritsky, I., "GIRS (Graph Information Retrieval System) Users Manual" (to be published).

(1) Dependence on disk I/O for pointer operations is reduced, which results in

   (a) reduced wall-clock time at the teletype terminal,

   (b) lowered over-all cost of program execution, and

   (c) the option of adding an efficient pointer traversal executive routine

(2) Multiple views of a data base are supported, so that the DBA can tailor the data structure to each application.

(3) A powerful Data-Definition Language/Data-Manipulation Language (DDL/DML) can be incorporated, which results in

   (a) greater convenience in querying, manipulating, and modifying data block relationships, and

   (b) greater convenience in inverting pointers to answer different types of queries

(4) Back pointers can be added automatically

These advantages are not without cost:

1. Implementation costs are incurred.

2. Some additional disk space is needed.

3. Disk space becomes fragmented for systems such as the CDC which have a fixed PRU size.

For the CDC 6700 computer, with its fixed PRU size of 64 words, the approximate number of PRU's needed for pointers can be determined by dividing the number of pointers in the entire data base by 64. For data bases composed mostly of data blocks longer than one PRU, the total amount of disk space required under COMRADE/GIRS may actually be only slightly different from that required under the COMRADE system.

A brief description of GIRS follows, after which the trade-offs among time, memory, disk space, and flexibility will be considered.

OVERVIEW OF GIRS (GRAPH INFORMATION RETRIEVAL SYSTEM)

ELEMENTS OF THE GRAPH STRUCTURE

GIRS is a hashed-address associative memory scheme designed to accommodate the insertion, retrieval, and deletion of information contained in arbitrary graph structures. Under this scheme, information is stored conceptually as primitive structures called node-link-node triples with the first node called the source node and the second node called the sink node, or value, as indicated in the following diagram.

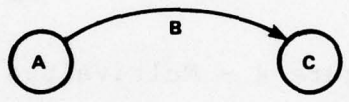
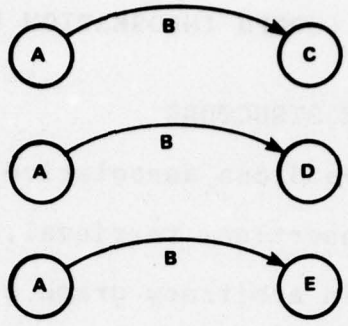


Figure 3 - Node-Link-Node Triple

Under a COMRADE/GIRS system, the source node A would represent the parent data block, the sink node C would represent the sibling data block, and the link B would represent the associating pointer or function.

The triple can, of course, be combined with other triples to form a more complex graph structure. The triple can be a component in a list (Figure 4).



alternatively drawn as

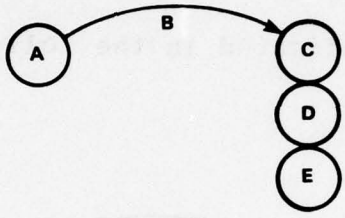


Figure 4 - Multivalued List

The triple can also be a component in a string (Figure 5).

alternatively drawn as

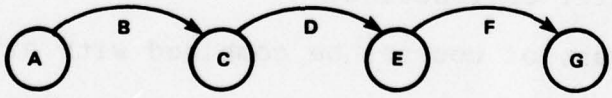


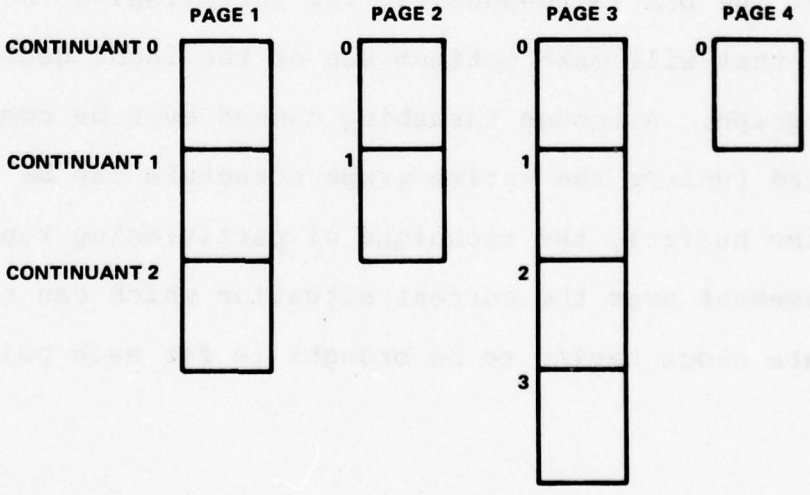
Figure 5 - Concatenated Value String

If the triple G,H,A were added to this string, each triple would then represent a component of a circuit.

PAGED (OUT-CORE) VERSION OF GIRS

There are three versions of GIRS currently available. Two are designed to use a working space or buffer which contains the entire graph structure in main memory. In one of these two, the buffer is divided into four separate arrays that store the node, list, conflict list, and flag functions; in the other, the four functions are packed into a single array. The third version, a paged (out-core) extension of the one which packs the functions into a single array, is the one that concerns us here. It is this version that is suggested for use with COMRADE.

In this out-core version of GIRS, the triple is assigned to a page that can extend its length as needed by a specified increment, called a continuant. These adjustable-length pages exist to enable information to be logically divided. Each page contains one or more logical records (continuants) of uniform length, as illustrated in the following diagram.



A continuant may be used either as a logical subdivision of the graph on that page or merely as an area for overflow from the previous continuant. Thus, although all of the pages have the same limit as to the number of source nodes they may contain, the sets of source nodes on the different pages are disjoint. Each continuant contains a subset, which may or may not be distinct, of the set of source nodes assigned to the total page. GIRS will automatically create continuants as needed.

By assigning an individual address to each continuant, the user can conveniently partition a graph into a number of "local" subgraphs. "Localness" of a subgraph may be defined in a number of ways. For example, all descendants of a node comprise a subgraph of "local" character. All or part of the nodes in a string or list may be considered local. To minimize thrashing (excess disk I/O due to a poorly designed data partition), the user must carefully partition the graph into local subgraphs which may be placed onto separate continuants. Note that the DBA is responsible for partitioning the graph in a way that will make optimum use of the local quality of a subgraph. Although thrashing cannot ever be completely eliminated (unless the entire graph structure can be stored within the buffer), the technique of partitioning represents an improvement over the current situation which can result in a new data block having to be brought in for each pointer access.

The following excerpt from Berkowitz[5] indicates some of the problems involved in relating the graph structure to the paged GIRS architecture. Consult Berkowitz[5] for a more detailed description of local graph processing.

> "The central problem is to determine on which page one should place a newly created node. If the program is creating a list of names of major graph localities, each new node should be on a different page than the current one, since... each page will have the capacity to contain at least the beginning of a graph locality. On the other hand, if the list is part of a hierarchical directory, then only the terminal nodes of the directory should be on different pages than the current one. Similarly, if the program is setting up strings to operate on a trace mode, then the newly created sink nodes should be on the current page or at worst on few enough other pages so that primary memory could contain all of them. More generally, graph manipulations are, in part, combinations of directory searches and string traces. While a directory search can result in a separate disk access to a new graph locality, it is most desirable that a trace take place within a given locality on one page. For example, consider a stack search which provides the information to continue a trace. In this case, the stack nodes are best represented as a list--i.e., a terminal directory--on a single page. The sink node addresses, however, are intended to represent links in a single graph locality, and hence are all used on the same page. If the link addresses are never used as source nodes, their place of definition is irrelevant."

The DBA also determines the continuant size and the maximum number of continuants to be resident in main memory at any one time. The length of the continuant determines the maximum number of unique source nodes that may be defined on a given page.

RELATIONSHIP OF DATA STRUCTURE TO GRAPH STRUCTURE

Under COMRADE, the POINTER is stored in the data BLOCK as a "data value". GIRS would represent this same relational information as triple (Figure 6).

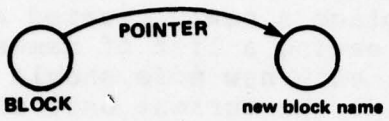


Figure 6 - GIRS Representation of a
Data Block Relationship

A relational triple such as that shown in Figure 6 would be packed into a single word of the GIRS buffer. Multivalue lists which can expand and contract in size dynamically would represent pointer arrays and would require one extra word of overhead just as at present under COMRADE.

Random values assigned by GIRS to BLOCK and POINTER (names assigned by the user) serve to place and locate the relationship within the GIRS buffer under a hashing technique. All new block names are assigned a random number along with a cross-file reference number if necessary. The cross-file reference number would associate a new block name with the appropriate COMRADE file in the data base. If the new block name is a hit, its random number is used in retrieving its Hollerith name which will be stored along with it in the GIRS buffer. The Hollerith name will then be given to the COMRADE Data Storage Facility

(CDSF)[7] which will bring the hit block into the COMRADE buffer. The procedure is discussed in greater detail later under the heading "Implementation".

The GIRS representation for both the intra-file pointers and the cross-file pointers would be identical. With a paged GIRS structure, a DBA could set up cross-file pointers and intra-file pointers for each file in the data base on a single page. This setup would be particularly suitable for a hierarchical directory. The primary effect of the shift of emphasis from the file structure to the page structure is that the Cross File Reference Table (CFRT) has to be modified. Under the present COMRADE system each file has its own CFRT. In a COMRADE/GIRS architecture, a single, universal CFRT for all of the files in a data base would be established, as will be discussed later in more detail.

TIME/SPACE/FLEXIBILITY/TRADE-OFFS, COMRADE/GIRS VERSUS COMRADE

DISK USE

Pointer Traversal under COMRADE

. COMRADE Dependence on Disk I/O

As already noted, under COMRADE the relational information about a COMRADE data block is physically a part of that block, so the block itself must be in main memory if the relationships of that data block with others are to be determined. Only a

---

[7] Bandurski, A. and M. Wallace, "COMRADE Data Management System - Storage and Retrieval Techniques," Presented at the 1973 National Computer Conference, New York (Jun 1973) American Federation of Information Processing Societies Proceedings, pp. 353-357.

very small portion of the data blocks of a COMRADE data base may reside in the working buffer at any one time (only 20 may be accommodated). Other data blocks not in the buffer may be needed to complete a traversal. To bring a block in, one, and sometimes two, disk accesses are needed.* The disk location of the desired data block is contained in a subdirectory, and when the subdirectory present in main memory is not the one that has the address of the particular block in question, the appropriate subdirectory must first be brought in. The data block is then accessed. As an example provided later will demonstrate, a pointer search involving more than just a few pointers will make extensive use of disk I/O under the COMRADE system.

. Cross File Pointer Traversal under COMRADE

It is in the area of cross-file pointer chasing that a COMRADE/GIRS combination is of the greatest value. Under COMRADE, a pointer chase at the QUERY language level can extend to only one other file at a time. Thus if the user has opened PERMFILE1, he may temporarily open and access PERMFILE2 but he may not go on to PERMFILE3. He must first return to PERMFILE1, after which he may then go to PERMFILE3. This procedure is both tedious and time consuming as the following sequence indicates:

---

* In the unlikely event that the size of the data block should exceed the size of the COMRADE I/O buffer and that the desired element should not be included in the retrieval portion of that data block, a third disk access would have to be made to obtain the relevant portion.

1. Close PERMFILE1
2. ATTACH PERMFILE2
3. Open PERMFILE2
4. Read from PERMFILE2
5. Close PERMFILE2
6. UNLOAD PERMFILE2
7. Open PERMFILE1

Depending upon the size of the load on the CDC 6700 computer system and the size of the file to be attached, the response time at the terminal might be quite slow.

Pointer Traversal under GIRS

. General Considerations

Under GIRS, pointer traversal is a relatively quick, flexible, and efficient operation. Moreover, since GIRS has been developed "in-house", it offers DTNSRDC users the added advantage of ease of maintenance.

One of the main reasons for selecting GIRS to handle COMRADE's pointer chasing would be for the reduction of disk I/O GIRS provides. Retrievals and insertions involving disk I/O are relatively slow compared to those performed in main memory and--due to the current cost of channel time--expensive.

No pointer retrieval/manipulation scheme can reduce the amount of disk I/O required for reading in "hit" blocks. If it were feasible to maintain the GIRS representations of the entire data structure in main memory at all times, disk I/O for pointer chasing would never be needed and only the in-core version of

GIRS, with its resultant lower space requirements, would be needed. However, since this is not always possible and out-core storage is sometimes needed, the use of disk I/O can at least be kept to a bare minimum by partitioning the data base. Somewhat less optimally, if the data base can be partitioned so as to combine on a single continuant all those relationships that are to be used together, a single disk read to bring that continuant into main memory will suffice. In this case, a simple, in-core GIRS retrieval would serve, instead of the several disk accesses ordinarily required to bring into main memory information which is mostly irrelevant. If more than one continuant is needed, the DBA must be clever enough to partition the data base so as to reduce the number of disk accesses required per query. In practice, it would be difficult to choose a partition so awkward that pointer traversals under GIRS/COMRADE would result in as many disk accesses per query being needed as under COMRADE alone.

The paged version of GIRS allows more than one page to exist in main memory at a time. Therefore, depending on how the workspace has been allocated, a continuant of a page may or may not have to be swapped into main memory. If GIRS should need the space occupied by a continuant, and should find that the continuant had already been modified when it was in main memory, the worst that could happen would be that the continuant would have to be written out to disk before a new continuant could be brought in. However, continuants are never modified as a result of a query. GIRS will allow a page to grow, continuant

by continuant, to absorb new portions of a graph. Note that if a retrieval fails--either because of the absence of a block name or because of a misspelled block name--all of the continuants of a page may have to be brought into main memory to be searched. Judicious choice of page size should minimize the number of continuants needed, and thus minimize GIRS I/O.

Part of the design philosophy of COMRADE is to maximize the use of a data block while it is in main memory and to thus forestall the need to bring back that same data block over and over again. The GIRS philosophy is similar. The graph, which represents the logical structure of the data, should be partitioned onto pages in such a way that the "localness" of the subgraphs will be preserved. In other words, whenever possible, subgraphs should be maintained on the same page and on the same continuant.

To illustrate, note that a graph can be broken up into lists and strings. A typical multivalued list might be structured as shown in Figure 7.

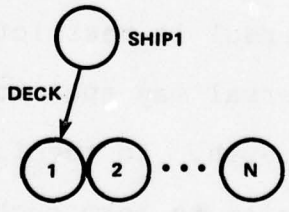


Figure 7 - A Typical Multivalued List

A string might assume the form of Figure 8.

Figure 8 - A Typical String

A group of strings may have a common source node, as indicated in Figure 9:

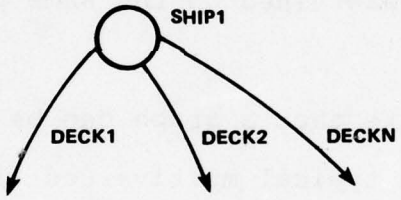


Figure 9 - A Collection of Length-One Strings
Having a Common Source Node

In some cases, traversal is restricted to strings or to lists. In others, traversal may span total subgraphs of a given string length and list depth. In the latter case, it is to the advantage of the programmer to keep such a subgraph on a single continuant.

. Cross-File Pointer Traversal under GIRS

As already mentioned, COMRADE allows only one file to be traversed at a time at the QUERY language level. Under GIRS, there is no such limitation. GIRS can handle cross-file pointers easily as intrafile pointers, since a universal Cross-File Reference Table would be set up and the cross-file reference number would be stored as part of every value (sink node) representing a data block. This feature would be beneficial for COMRADE applications involving distributed data bases.

Under GIRS, the pointer to data records in various different files may be traversed by a single query, using little if any disk I/O, since cross-file and inter-file pointers may be mixed on the same GIRS page. This capability can be assured by collecting all of the necessary pointer relationships onto a single or a relatively small number of continuants. Of course, severe thrashing can result if the necessary relationships of a subgraph are scattered across many continuants, most of which are not in main memory at the time they are needed. It is up to the DBA to make sure that the graph structure is thoughtfully prepared.

RESPONSE TIME

Response Time of a COMRADE Retrieval

A COMRADE retrieval is heavily dependent on disk I/O, which can consume a lot of wall-clock time. This dependence needs to be reduced. COMRADE permanent files are stored on both the 841 and 844 disk drives for the CDC 6700 computer system. An I/O

access to a CDC 844 disk drive will generally take between 40 and 1000 milliseconds, depending upon the amount of time lost while waiting for an available I/O channel. Although statistics on the average disk I/O service time for the CDC 6700 computer system are not available, those for the CDC 6400 computer system indicate that the average disk I/O service time for the CDC 6400 computer ranges between 200 and 280 milliseconds. The average disk I/O service time for the CDC 6700 computer is thought to be comparable. Average access times in milliseconds for the different disk drive units are provided in Table 1.*

TABLE 1 - CDC DISK DRIVE SERVICE TIME REQUIREMENTS

(in milliseconds)

	CDC 841	CDC 844
Average positioning time	69.5	30.0
Latency time (50% of rotation time)	12.5	8.4
Time to read one sector or PRU	3.6	1.4
Channel wait time (range)	0.0 - 960.0	0.0 - 960.0
Total service time (range)	85.6 -1046.0	40 -1000.0

* This information was obtained from Kenneth C. Rieck of the Center.

Using the "Presidents" Data Base[8] (described in Appendix A) and operating from a 10-character-per-second teletype terminal, a COMRADE query of the form

PRINT STATE, CAPITAL .OF. HEAD/ADMIN/PRESPTR/ADMIN/STATEPTR/

required approximately 230 data block accesses for pointers. On two tries, this query took 7-8 minutes. On an earlier occasion, involving an older model CDC disk drive system, the query took 13 minutes. Fifty of the 230 data blocks retrieved were hits, while the remaining 180 were extraneous. The buffer size was set to 1025 words, which allowed as many as 16 data blocks to reside in main memory at a time. A table has been prepared which shows, for N = 2 subdirectories, the probabilities of the following situations and the number of disk accesses needed for each. (Of course, these probabilties assume a random distribution of the blocks to be retrieved):

1. The desired data block is already in the buffer.
2. The data block is not in the buffer but the appropriate subdirectory is.
3. Neither the desired data block nor the appropriate subdirectory is in main memory.

---

[8] Willner, S. et al., "COMRADE Data Management System," Presented at the 1973 National Computer Conference, New York, (Jun 1973) American Federation of Information Processing Societies, 1973. pp. 345-349.

TABLE 2 - PROBABILITY OF A DATA BLOCK OF THE "PRESIDENTS" DATA BASE RESIDING IN MEMORY

Contained in Main Memory	Number of Disk Accesses Needed to Retrieve Desired Data Block	Probability of Situation
Situation 1: Block	0	$P_B = (1024/64)/281 = .057$
Situation 2: Subdirectory but not block	1	$P_S = (1-P_B)/N = .471$
Situation 3: Neither block nor subdirectory	2	$P_{neither} = (1-P_B)(N-1)/N = .471$

The number of disk accesses likely to be needed per data block retrieval is computed as follows:

$$E = P_S = 2(1 - P_S - P_B) = 2 - P_S - 2P_B$$

where P_B is the probability that the data block is in main memory

$$= \frac{\text{buffer size/average data block size}}{\text{total number of data blocks in data base}}$$

P_S is the probability that the subdirectory is in main memory

$$= \frac{1 - P_B}{\text{Number of subdirectories}}$$

For the "President's" Data Base, E = 1.4146. Thus there are 180E = 255 unnecessary disk accesses. I/O time is computed as follows: T = number of data blocks retrieved times E times the disk I/O service time. For the "Presidents" Data Base, T ranges between 1.08 and 1.52 minutes. The time required

for the equivalent operation in a COMRADE/GIRS scheme is discussed in the next section.

One final remark. P_B was computed on the basis of random allocation of block names to the subdirectories. If the user were allowed to specify to the lower level CDSF routines that certain collections of data block names should be kept on the same subdirectory, P_B could be greater and both E and T could be reduced. The argument for this approach is analogous to that for partitioning the data block representation in GIRS, and will not be pursued further.

Response Time of a GIRS Retrieval

So far, there have been no timing tests performed on the out-core version of GIRS, although the in-core packed-word version has been tested. In the in-core version, a simple retrieval for a triple (which might contain pointer or other information) requires 78 microseconds, a magnitude three orders faster than that required to access the 844 disk drives. Other retrievals take up to 142.6 microseconds (worst case).

Average insertion, deletion, and retrieval times for the present unpaged and unpacked version of GIRS are documented in Berkowitz.[4] Assuming that the proper page-continuant already exists in main memory, similar operations under the out-core version might reasonably be expected to take slightly longer, since relative addresses within the GIRS buffer would have to be resolved.

The disk I/O time for a query under COMRADE/GIRS (T_G) could be determined as follows:

T_G = (number of hit blocks) x E + (number of GIRS continuants brought in) x (disk I/O service time)

In the COMRADE example involving the "Presidents" Data Base (given in the previous section), there were 626 pointers and 274 data blocks.* Two hundred thirty data blocks were read in to obtain the 50 desired data blocks (180 data block retrievals were extraneous). If GIRS were to be used along with COMRADE, this entire data block representation could be contained within main memory, thus obviating the need for any I/O for GIRS. If, for some reason, the entire data base did not fit within main memory, disk I/O would be needed to bring in the necessary continuants. Even so, the total amount of I/O needed would be substantially less than that required under COMRADE alone. The speed advantage of GIRS is extremely query-dependent and would be even greater if more non-hit blocks were involved. The speed advantage for a single query is expressed as

$$\frac{D}{(H + C/E)}$$

where D is the total number of data blocks brought into the buffer in the present system

H is the number of hit blocks for the query

C is the number of GIRS continuants that must be brought into main memory

---

* These figures were supplied by Stanley E. Willner, developer of the "Presidents" Data Base.

For this example, the speed advantage would be D/H or (230/50) or 4.6, since the entire relational structure would fit entirely within main memory.

UTILIZATION OF DISK SPACE

Data Block Size, Percentage of Pointers per Data Block, and Fragmentation

The SCOPE disk operating system can handle words only in 64-word groups, these groups known as physical record units (PRU's). As a result, any record written out to disk must take up some multiple of 64 words. All data records of a size from 1 to 63 words (not counting the end-of-record marker) would require the same amount of space (64 words), and those of 64-127 words would take twice that amount of space, and so forth. Thus a great deal of fragmentation--unused but available space--could result.

At the present time, COMRADE users work approximately 90 percent of the time with blocks containing from 20 to 50 words.* Thus, so far as space allocation is concerned, it would be of no great advantage to them to have the size of these data blocks reduced by removing pointers. However, future plans for the use of COMRADE envision the need for data blocks larger than one PRU. The question, then, is: At what point does it become advantageous spacewise to compress the data blocks by removing the pointers and assembling them elsewhere on a file of their

---

* Data obtained from Bernard M. Thomson of the Center.

own or at the end of the data file? (Other reasons for separating the pointers are discussed elsewhere in the section entitled "Time/Space/Flexibility Trade-Offs, COMRADE versus COMRADE/GIRS.")

In the current COMRADE scheme, each data block relationship takes up one word of storage plus any space needed for pointer names in the BTD tables. In a COMRADE/GIRS setup, one word is needed for each data block relationship represented by a node-link-node triple. However, additional space is needed to link the GIRS random number representing the sink node with its Hollerith block name and vice versa, since CDSF cannot use the GIRS random number, and the block name must be converted to a unique random number to be operated on by GIRS. Conversion of the random number to the block name requires one or two additional words per name, depending on the length of that name.

To convert the Hollerith name to its associated random number, at least one additional word per block name is necessary; however, the exact amount of space required varies. The less the correlation among the block names, the greater the amount of space required. Thus, the three names EQUIP1, EQUIP2, and EQUIP3 will require less space than the names EQUIPMNT, MACHINES, and MOTORS. Block names in COMRADE data bases generally exhibit a high degree of correlation. This will be discussed in further detail in the section entitled "Implementation".

For programs needing successive runs, additional space may be needed under COMRADE/GIRS to store the values of nodes and links, depending upon which of two methods is used. In the one method, all of the nodes and links are written out to disk at the end of the program and then read back in at the beginning of each run; in the other, values for the nodes and links are reassigned by the GIRS pseudo random number generator at the beginning of each run (the same sequence of values will be used), so no extra storage space is needed.

The disk space allocation under the two different systems is computed as follows:

Under the COMRADE system:

$$\text{Total number of PRU's} = M\left(\left[\frac{w}{64}\right] + 1\right)$$

Under the COMRADE/GIRS System:

$$\text{Total number of PRU's*} = M\left(\left[\frac{w(1-P)}{64}\right] + 1\right) + C\left[\frac{Mwp/C+h}{64}\right] + 1 + \left[\frac{2M}{64}\right] + 3$$

where M = number of data blocks

w = average number of words/data block

p = average percentage of pointers

[] = greatest integer function (i.e., $3.8 \rightarrow 3$)

c = total number of continuants for all pages

h = nine header words/continuant

---

* This computation includes the space needed to store data block names.

From the expressions just given, one can see that under the proposed system a data base would require slightly more space than under COMRADE, except in those cases in which the data blocks would be one PRU in length. If, however, a computer system having variable length PRU's (such as the IBM disk operating system) were to be used, the ratio of COMRADE disk space to COMRADE/GIRS disk space would be approximated as follows:

$$\text{Disk Space Ratio (DSR)} = \frac{\text{COMRADE/GIRS disk space*}}{\text{COMRADE disk space}} = 1 + \frac{2M+9C}{Mw}$$

$$= 1 + \frac{2}{w} + \frac{9c}{\text{total number of words in orig. data base}}$$

Let us now determine the range of this ratio for a ship described in a mature data base of the Ship Design File. A typical data base in this case might consist of 40 to 60 percent pointers. We have chosen a very conservative working space/available-space ratio of 7 to 3. Furthermore, as an upper bound on term $2/w$ of the DSR, the number of elements to be included per data block has been set at nine. Since the latter number and the given GIRS ratio are unrealistically low, values thought to be more typical will also be considered. For example, the average number of elements per data block might more reasonably be 30. Thus, for an average of 9-30 elements per data block, the term $2/w$ would then range from .22 to .07. Table 3 gives the DSR for six different cases.

---

* This number includes the space needed to store the data block names.

TABLE 3 - DISK SPACE RATIO (DSR), COMRADE/GIRS TO COMRADE
(Quantities expressed in number of words)

COMRADE	Number of attribute elements	66000		66000		66000	
	Number of pointer elements	44000		66000		99000	
	Total number of elements in data base	110000		132000		165000	
	Percentage of Pointers	40		50		60	
GIRS		*	**	*	**	*	**
	Continuant size	100	25	100	25	100	25
	Percentage of GIRS buffer being used (not in Available Space)	85	70	85	70	85	70
	Required GIRS buffer space	51725	62858	77647	94286	116470	141429
	Number of continuants needed	518	2515	777	3772	1165	5658
	Third term of DSR	.04	.21	.05	.26	.06	.31
	Minimum DSR	1.11	1.28	1.12	1.33	1.13	1.38
	Maximum DSR	1.26	1.43	1.27	1.48	1.28	1.53

* Typical case
** Conservative case

As Table 3 shows, the third term of the DSR equation ranges from .04 to .31. Therefore, the DSR ranges from 1.11 to 1.53, with 1.11 the more typical.

GIRS Continuant Size and Unused Entry Space

The continuant size must be determined in advance by the DBA. The hashing scheme employed by GIRS results in the size of the continuant determining the maximum number of identifiers or node and link names that can be created for a page. All of the continuants throughout the GIRS structure of a given data base will be of the same size.

Under GIRS, the use of many small continuants rather than a few large continuants for a given size graph structure might result in the need for more disk I/O, since more searching might be needed to find a requested relationship. However, using more but smaller-sized continuants could result in a fuller use of the entry space in GIRS. If the continuants are used merely to store overflow, only the last continuant of each page will have any unused entry space. Moreover, with the GIRS hashing scheme, any partially empty continuants will have what are known as short conflict lists which facilitate quick retrieval response.

Example of Disk Use under COMRADE and under COMRADE/GIRS

The following example compares the time and space requirements under COMRADE with those under COMRADE/GIRS. This example illustrates that, when the search path of a pointer chase exceeds one level, COMRADE/GIRS can perform the query in less time than COMRADE at a cost of only slightly more disk space.

In the nomenclature of COMRADE, assume three pointer arrays DECK(L), COMPART(M), and EQUIP(N)--with the respective lengths

L, M and N. The pointer array COMPART(M) is repeated in L different data blocks and the pointer array EQUIP(N) is repeated in LM different data blocks, resulting in LMN number of hits. The following query is made:

PRINT COST .OF. SHIP1/DECK/COMPART/EQUIP

Assume also 1000 data blocks and six subdirectories. Under the COMRADE system, only one subdirectory may be in main memory at a time. The minimum block size is 64 words, and the COMRADE I/O buffer is set at 1281 words. Therefore, as many as 20 data blocks may be in main memory at any one time. The following table indicates the probability of the desired item residing in main memory and the number of disk accesses involved:

TABLE 4 - PROBABILITY OF A DATA BLOCK OF THE TIME/SPACE *EXAMPLE RESIDING IN MAIN MEMORY*

Possible Situations in Main Memory	Number of Disk Accesses Needed to Retrieve Desired Data Block	Probability of Situation
Situation 1 Block	0	P_B=(1281/64)/1000=.02
Situation 2 Subdirectory but not block	1	P_S=(1-.02)/6=.1633...
Situation 3 Neither the block nor the sub-directory	2	$P_{neither}$=(1-.02)(5/6) = .8166...

The number of disk accesses expected per data block retrieval E for any one data block would be expressed as

$$2 - .1633...-(2 \times .02) = 1.7966...$$

Under COMRADE, this query would result in E(1+L+LM+LMN) number of disk accesses where $E(0 \leq E \leq 2)$ is the expected number of disk accesses per data block.*

The actual number of disk accesses under COMRADE might even be greater, since the COMRADE buffer may contain only 20 data blocks at a time, forcing certain data blocks to be brought into main memory more than once.

In the GIRS scheme, the contents of each pointer array are represented as a multivalued list. The data block names are assigned unique "random" numbers which are represented by dollar ($) signs. For example, SHIP1 has the value "$\$_1$". The COMRADE pointers DECK, COMPART, and EQUIP become GIRS functions or links. The links must also be assigned unique "random" values. The GIRS pointer graph would have to contain the following structure to handle the query of the example:

---

* In the unlikely event that the data block size should exceed the buffer size and the desired element were not present in the retrieved portion, then $0 \leq E \leq 3$.

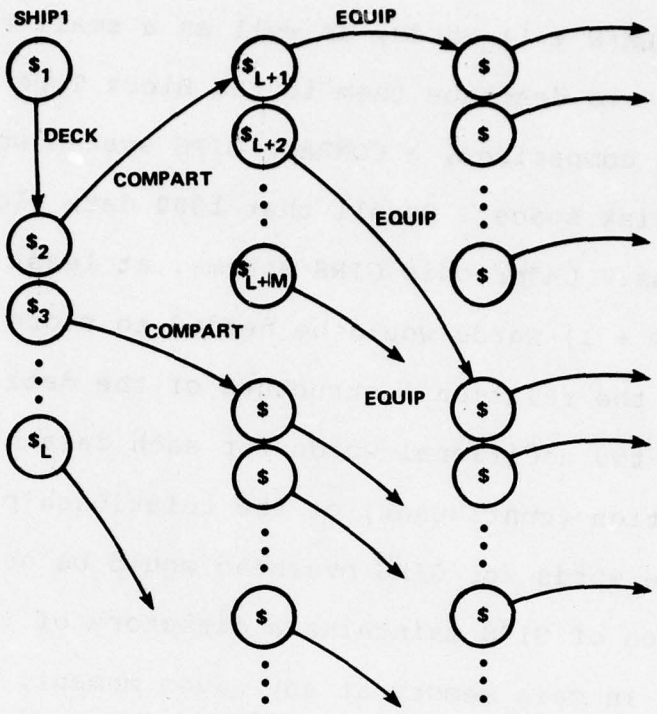


Figure 10 - GIRS Structure for Example

Using COMRADE/GIRS, the query given would result in L + LM + LMN GIRS retrievals to traverse the graph, and LMN retrievals for the Hollerith representations of the block names. (Conceivably, all of the nodes could be contained within main memory, resulting in no need for disk I/O.) The CDSF routines would be given LMN hit block names which would result in E(LMN) disk accesses, thus eliminating the need for E(1 + L + LM) disk accesses.

Clearly, there is a potential time--and, therefore, cost--savings under COMRADE/GIRS, due to the reduced need for disk I/O. On the average, the time savings would range from .36(1 + L + LM) to .5(1 + L + LM) seconds.

Under the present COMRADE system, the pointer arrays require $L + L(M + 1) + LM(N + 1)$ words, as well as a small amount of additional space to describe them in the Block Type Definition (BTD) file. In comparison, a COMRADE/GIRS system would require somewhat more disk space. Recall that 1000 data blocks exist in the data base. Under this GIRS scheme, at least $(L + 2) + L(M + 1) + LM(N + 1)$ words would be needed to store the graph that describes the relational structure of the data base, as well as one or two additional words for each data block name. For each partition (continuant) of the relationship graph, an additional nine words for GIRS overhead would be needed. The out-core version of GIRS maintains a directory of the continuants that are in main memory at any given moment. The directory size, which is determined by the applications programmer, is set at one more than the first multiple of 64 that is greater than the number of in-core continuants. A copy of the directory is stored on disk at the start of the program. In general, the directory will take up only one or two PRU's. For example, say that GIRS used the COMRADE buffer (1281 words) for its own buffer. Even if the continuant size were set at only ten words, the directory size d would be

$$\frac{\left(1281 - \left(\left[\frac{d}{64}\right] + 1\right) 64\right)}{10} = 64\left(\left[\frac{d}{64}\right] + 1\right) - 1$$

$$= 128 \text{ words or two PRU's}$$

Let us now determine how much extra disk space would be required under COMRADE/GIRS for the same graph. If L, M, N are 20, 30, and 40 respectively, and if the continuant size is set to 100, the directory will require one PRU or 64 words. Let us also assume a 15 percent available space in the GIRS buffer. This graph would then require a pointer space of 29697 words, requiring 297 continuants. Accordingly, the COMRADE/GIRS system would take 1 + (9 x 297 + 200 + 64)/25242 words of storage, or 1.188 as much disk space as under COMRADE, plus .15 of that amount to account for the available space thereby totaling 1.388. Of course, if this "data base" contained attribute data, that ratio would be lower. For example, 50 percent attribute data would bring the ratio down to 1.169.

CORE REQUIREMENTS FOR COMRADE/GIRS

The out-core version of GIRS can be conveniently separated into two parts, a major part and a minor part. The major part, which takes up 7350_8 (3800) words plus an additional 110 words for APAGE, a CDSF routine, performs all of the main functions. The GIRS buffer, which contains the in-core portion of the data block relationship graph, would not use up any additional main memory since the COMRADE buffer and the GIRS buffer would share the same memory space. If the DBA could be certain that the relational information would fit entirely within the GIRS buffer, only the in-core packed version of GIRS, which takes up about 4300_8 (2250) words, would be needed. Even if GIRS were split up with about 3340_8 (1760) words going to the major portion,

the major functions could still be performed. If only querying of the relational structure were to be performed, only 370_8 (250) words of the GIRS package would be needed. A query using the out-core version of GIRS would still require 7350_8 words.

GIRS is a subroutine package which requires an executive routine to create the subroutine calls. It is similar to Subroutine QUERY in this respect. Subroutine QUERY handles a parsed ".WHERE." clause, whereas this new executive routine would be required to handle a parsed ".OF." clause. Since an executive routine has not yet been written, we cannot accurately gauge its size; however, a program for using GIRS interactively at the terminal already exists which takes 1060_8 (560) words. This program has functions similar to those needed by an Executive routine.

The amount of CDC main memory available to an applications programmer is limited to $60K_8$ (24,600) words when a remote terminal is used. Currently, $27K_8$ (11,800) words are allotted for COMRADE functions, although plans are being made to reduce this size to $20K_8$ (8200) words in the near future. Also, approximately 1K is used for the COMRADE I/O buffer.

Since COMRADE is being revised, information is not yet available as to the amount of main memory to be required. From the numbers just discussed, however, we can state that, at worst, GIRS will require $10,400_8$ (4360) words. COMRADE developers feel that COMRADE/GIRS will fit within the 20K constraint.

FLEXIBILITY

Modifying Data Block Relationships

A COMRADE/GIRS architecture would provide a great deal more flexibility in handling data. Such a system would enable the DBA to create, update, and remove all or any part of a description of the data base with relative ease. The data definition dictionary could be created and modified independently of the data base itself, so there would be no necessity for moving data records into and out of main memory.

Pointer Traversal Executive Routine

Currently, COMRADE allows the data base to be queried at two different levels - namely, via the CDMS subroutines and via the QUERY processor - which provides the user with flexibility/convenience trade-offs. The latter level accepts English-like requests, parsing and converting them into CDMS subroutine calls. Although COMRADE already has an inverted query executive routine (called QUERY), it does not have a pointer-traversal executive routine. The advantage of such a routine for COMRADE may be illustrated by a discussion of the way in which COMRADE presently must handle the following three different queries.

1. PRINT BN .WHERE. AREA .EQ. 20;
2. PRINT COST .OF. BLOCK1/PTR1/PTR2;
3. PRINT HEIGHT .OF. BLOCK2/PTR3/PTR4 .WHERE. AREA .GT. 1000;

In the first instance, and using the higher level query procedure, AREA must be on an inverted list. The .WHERE. clause

will be parsed and then the equivalent code to Subroutine QUERY (which is part of the QUERY processor) will be called to bring in and search the inverted lists and then return the names of the data blocks having AREA = 20. These blocks are the "hit" blocks. The query could also have been handled in COMRADE at the lower level with the applications programmer calling a COMRADE executive subroutine (also called QUERY) directly, supplying input parameters in the equivalent of a parsed .WHERE. clause. This method, which is convenient to use, would allow an applications program to use the output of the Subroutine QUERY immediately.

In the second instance, the QUERY processor must call a series of routines to bring in BLOCK1 and the data records associated with PTR1 and PTR2. The applications programmer, in order to have greater flexibility (for example, to be able to use the retrieved data immediately or to restrict the pointer chasing to blocks found in the nth repeating group), will have to make several subroutine calls for a typical pointer chase query, since there is no pointer-traversal executive routine currently in COMRADE analogous to Subroutine QUERY to handle the parsing of an .OF. clause. Under the present system of COMRADE, use of a pointer chasing routine would be inefficient, since hit blocks would have to be removed and later brought back into main memory when the buffer was filled. A pointer executive routine under a COMRADE/GIRS combination would accept the equivalent of a parsed .OF. clause

as input. The first input parameter (the starting data block) would be treated as a source node and the rest of the input parameters (the pointer names) as link names. Output would be a list of (potential) hit blocks. Therefore, the number of hits per query would then be available. This same type of query will be discussed for the data definition/data manipulation (DDL/DML) language level in the section on adding a DDL/DML to COMRADE.

In the third instance, the pointer list is traversed as in the second instance, resulting in a sequence of potential hits. Each of these potential hit blocks is brought in (along with the non-hit blocks needed for the pointer traversal) at which point the proper attributes are tested against the conditions of the .WHERE. clause to determine whether or not the block is truly a hit.

As already mentioned, most of the disk I/O needed under COMRADE for retrieving the non-hit blocks would be eliminated under the COMRADE/GIRS scheme. In addition, disk I/O use could be reduced even further by including a pointer executive routine within a COMRADE/GIRS system. Such a routine would bring into main memory only those potential hit blocks which could ultimately satisfy the .WHERE. conditions. It would accomplish this by inverting the conditional elements involved in a query composed of both .WHERE. and .OF. clauses, and by then ANDing the potential hits listed by the pointer executive routine with the potential hits of the output list of the .WHERE. clause.

It seems clear that some kind of a CDMS-level routine is needed which could efficiently handle a parsed .OF. clause. The COMRADE/GIRS combination could satisfy this need, as is explained later on in the section entitled "Implementation."

Adding a Data Definition/Data Manipulation Language to COMRADE

Basically, a Data Definition Language (DDL) is used to create the data base structure, whereas a Data Manipulation Language (DML) is used to access and modify the data base. The necessary and optional characteristics of DDL/DML's are detailed in Martin[9] and by the National Bureau of Standards.[10] A COMRADE/GIRS system would enable the introduction of a concise, powerful DDL/DML, since GIRS is designed for pointer manipulation. Structural relations described by a DDL/DML would be easier to visualize and therefore easier to work with. Because the one-to-one correspondence between the pointer relationship and its DDL/DML code would simplify the conversion of the conceptual relational graph from paper to a DDL/DML computer program, programming time would be reduced. Further, a DDL/DML would allow the user to more easily write subroutines capable of performing such functions as deleting a block and all of its descendants or connecting

---

[9] Martin, J., "Computer Data Base Organization," Prentice-Hall, Inc., New Jersey (1975), pp. 100-1.

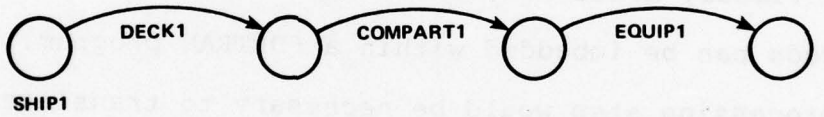
[10] "CODASYL Data Description Language," U.S. Department of Commerce, NBS Handbook 113, (Jun 1973), pp. 2.11-2.12.

a parent of a block to the sons of that block when that block is to be deleted.

A particularly convenient DDL/DML already in existence is GIRL.[11] The abstract from the GIRL Programming Manual follows:

> "GIRL (Graph Information Retrieval Language) is a programming language designed to conveniently manipulate information in graph structures. As such, the language will play a key role in the construction of the organizational schemes found, for example, in information retrieval, pattern recognition problems, linguistic analysis, and process scheduling systems. The language is written to complement an algebraic language, in the sense that GIRL statements are distinguished from the statements of the algebraic language and the statements may be interleaved. The primary advantage of separating symbolic and numeric statements is that the programmer is afforded a linear, one-one trace of graph operations in the code description."

For example, using GIRL to create or add to the following structure shown originally in Figure 8,



the following code

```
G    SHIP1   DECK1   $   COMPRT1   $   EQUIP1   $
```

would suffice. To find the EQUIP1 data block of SHIP1, the code

```
G    SHIP1 + DECK1 + COMPRT1 + EQUIP1
```

might be used. Modification and deletion would be equally straightforward. Further examples are provided in the section, "Pointer Operations under COMRADE and under COMRADE/GIRS".

---

[11] Berkowitz, S., "Graph Information Retrieval Language; Programming Manual for FORTRAN Complement," Naval Ship Research and Development Center Report 4137 (Jun 1973).

Note that, in the COMRADE QUERY language, the ".OF." list for a single query is limited to eight distinct levels of pointers and a total of 16 pointer accesses.* This limitation is built into COMRADE to prevent a pointer chase from continuously looping on a circuit. Of course, at the CDMS level, even under the present COMRADE setup, an applications programmer may first create temporary pointers to tag data blocks as having been visited and them remove these pointers, although this procedure would result in an even heavier use of disk I/O and would require the applications programmer to generate several CDMS subroutine calls. Under GIRS, and with the use of a DDL/DML such as GIRL, it would be a simple matter to temporarily tag a node (data block) or thread a subgraph. The previously mentioned limitations would no longer apply. GIRL code can be imbedded within a FORTRAN program. Although a preprocessing step would be necessary to translate GIRL to FORTRAN, a GIRL preprocessor already exists which is portable and currently operational on the CDC 6700 and the PDP-11/45. Moreover, a GIRL/FORTRAN compiler is already being written for the PDP-11/45.

A single GIRL statement can take care of either a single CDMS pointer operation or a series of CDMS pointer operations that ordinarily require several lines of code (See Appendix C).

---

* COMRADE, however, does not permit the pointer name of a level to be tacitly repeated if the sink block contains the pointer, giving rise to a tree of pointer strings.

Back Pointers

There are other advantages to the COMRADE/GIRS combination. One is the ease of adding back pointers. This could be done at the DDL/DML level with the following statements:

```
C    ADD POINTER RELATIONSHIP
G    A    B    C
C    ADD BACK POINTER
G    C    B    A
```

These two GIRL statements can themselves be combined

```
G    A    B    C    B    A
```

By setting a mode flag in the applications program to .TRUE., the programmer could even cause back pointers to be created within the pointer executive routine automatically. This automatic creation would continue until the flag is changed to .FALSE.

Pointer Inversion and Query Type Specification

The preceding discussion has shown that not only the question A B ? but also the question C B ? can be answered. What about the questions ? B C or A ? C. These queries might translate to "What objects have DECK1?" and "How are ships and decks related?" A COMRADE/GIRS scheme would handle these questions by hashing (C,B) and (A,C). A flag would be set to indicate the appropriate query type.

At the cost of creating, maintaining, and storing some ring structures, the queries A ? ?, ? B ?, and ? ? C could also be satisfied. This topic is discussed by Berkowitz.[5]

Storage of Nonpointer Data

GIRS not only has the ability to store and manipulate pointer relationships, but also to store integer and Hollerith data. Numbers as large as 2^{18} and with as many as three characters may be stored directly within the word in the buffer that holds the triple. This capability would be valuable to a programmer who wanted to associate a particular relationship with a subgraph level. Hollerith and integer data exceeding the stated limits could be stored in a separate, sequentially allocated buffer called SEQSPC. This arrangement provides space for the rapid retrieval of data such as data block names.

Satisfying an Imprecise Query

Under COMRADE, an engineer wishing to locate attribute data stored several levels below his starting block must have knowledge of the pointers at each and every level involved in the pointer traversal. All of the pointers must be specified to satisfy his query. This situation can be circumvented, at the expense of some software design and computer time and space, by having the DBA provide a generic description of the data structure as well as the actual data structure, as illustrated by the following example.

Let us say that an engineer wishes to query the structure of Figure 11.

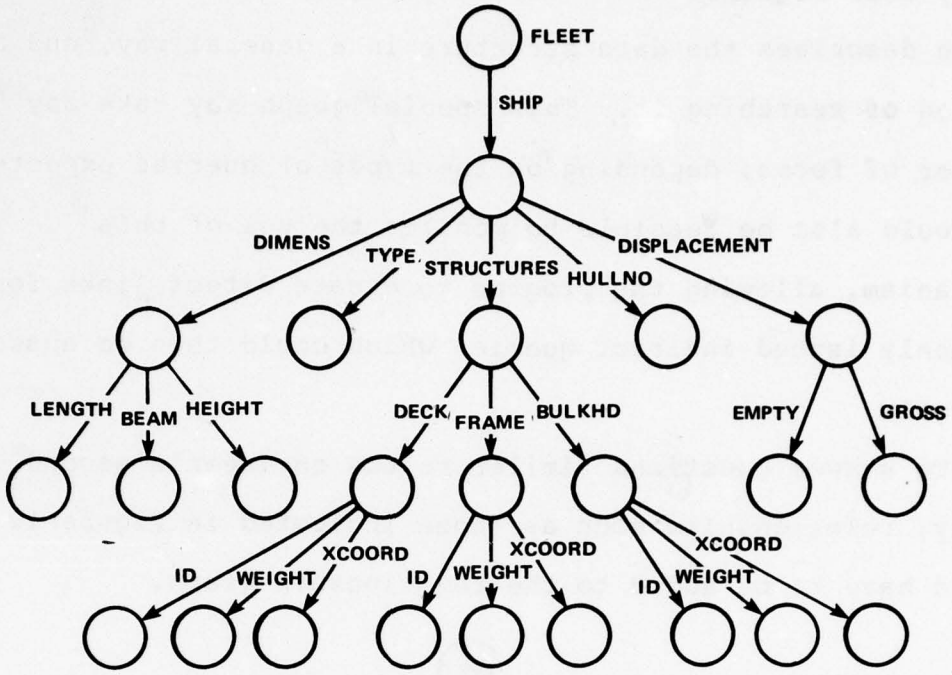


Figure 11 - Ship-Design Data Structure

He may ask, "What types of structures do ships have?" This query is easily answered, since "structures" are only one level below "ships" in the graph. This is known as a direct query.

But what if he wishes to ask "Which decks do ships have?" This query could not be satisfied at present, since "deck" is more than one level below "ships" in the graph. Since "ship" and "deck" are not immediately related, this query would be called indirect. An indirect query could be handled by allowing the GIRS retrieval routine (FIND) to use the knowledge of the different types of permissible relationships in the graph. Aside from the data base itself, an indirect

query also requires the relationship graph and a special graph which describes the data structure in a general way, and a method of searching it. This special graph may take any number of forms, depending on the types of queries expected. It would also be feasible to monitor the use of this mechanism, allowing the program to create direct links for commonly issued indirect queries which could then be answered directly.

To answer questions similar to the engineer's second query, relationships such as those indicated in Figure 12 would have to be added to the relationship graph.

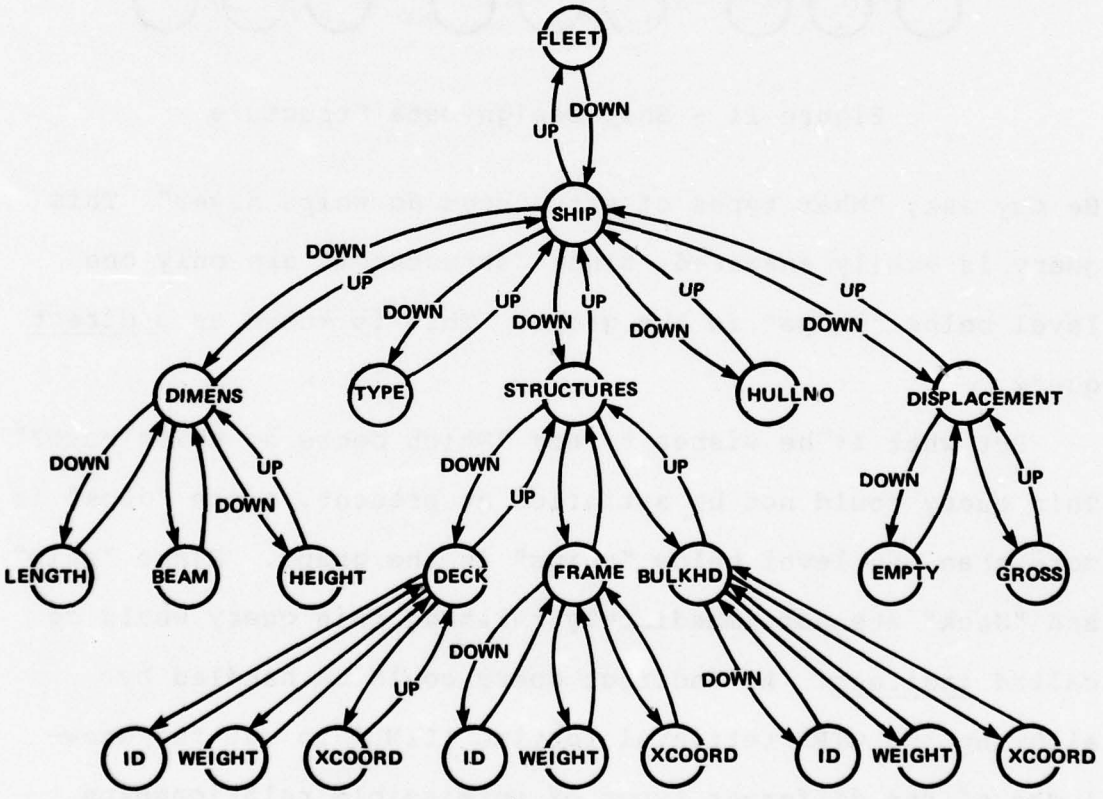


Figure 12 - Generic Description of Ship Design Data Structure

Under COMRADE/GIRS each user would design his own search mechanism, to be called by FIND when a direct query fails. If he did not wish to do so, his query would be unsuccessful. An alternative but possibly less efficient way of handling this problem would be to permanently embed a general, fixed strategy into the system which would require only that the applications programmer submit a list of relationships. Such a strategy might perform a level-by-level, breadth-first search of the graph, as indicated by the flowchart of Figure 13, taken from Berkowitz.[5]

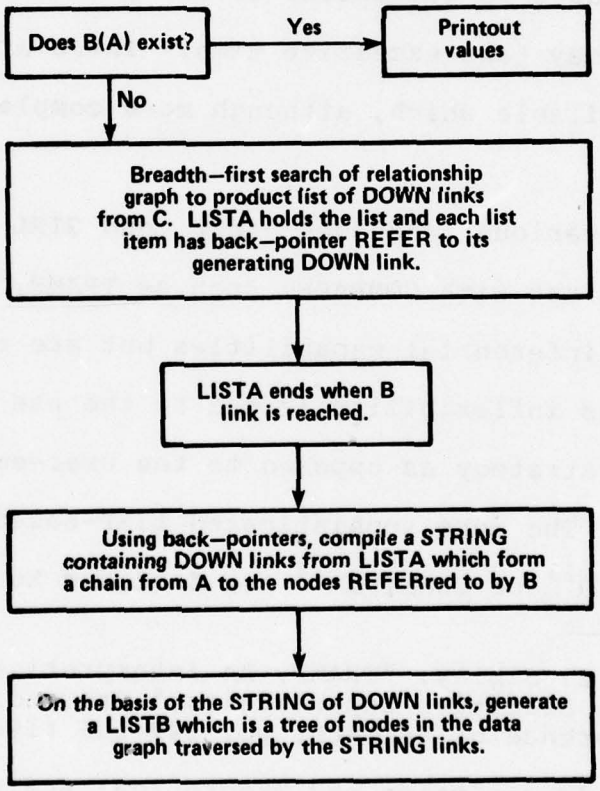


Figure 13 - Search Procedure for an Indirect, Memory-Related Query

The GIRL/FORTRAN code, also from Berkowitz,[5] is included in the Appendix. Note that the retrieval--or, for that matter, the insertion or deletion--strategy is essentially embedded in GIRS, but may be written in GIRL (as has been done in Appendix B) for descriptive convenience.

To apply the algorithm of Figure 13 to the engineer's indirect query, the variables A, B, and C might represent ships, decks, and the next level of nodes retrieved, respectively. DOWN, STRING, and REFER are used as reserved GIRS link identifiers, and LISTA and LISTB are reserved GIRS node identifiers.

This procedure is guaranteed to work, but such a "brute force" method may take excessive time. There are many other procedures available which, although more complex, are more powerful.

There are various languages other than GIRL which might be considered for use with COMRADE, such as TRAMP,[12] for example, which possess inferential capabilities but are relatively inflexible. This inflexibility is due to the use of a fixed predetermined strategy as opposed to the user-embedded strategy of GIRS/GIRL. The more sophisticated LISP-based languages such as PLANNER[13] or CONNIVER[14] are designed to handle complex

---

[12] Ash, W. and E. Sibley, "TRAMP, An interpretive Associative Processor with Deductive Capabilities," Proceedings of the 23rd National Conference of the ACM, pp. 143-156 (1968).

[13] Hewitt, C., "Description and Theoretical Analysis (using schemata) of PLANNER," MIT Artifical Intelligence Laboratory AI-TR-258 (1972).

[14] McDermott, D.V. and G.J. Sussman, "The CONNIVER Reference Manual," MIT Artificial Intelligence Laboratory AIM-2599 (1974).

strategies but are not interfaced with a numeric language like FORTRAN or ALGOL, and are relatively non-portable. On the other hand, they may indeed be suitable candidates for a DDL/DML if the prime emphasis is to be on the inherent "intelligence" of the data management system. For a system with modest inferential capability and relatively low overhead, however, the COMRADE/GIRS combination seems preferable.

There are other types of indirect, possibly inverted, queries which the engineer may wish to submit, as for example,

What objects have the structure A?

What ships have the structure C?

How are ships and decks related?

A different algorithm would be required for each of these queries.

In conclusion, the search mechanism of the proposed system is a powerful tool which would provide the user of COMRADE with a great deal of flexibility, even though it would require more memory and additional computing time.

THE ROLE OF THE DATA BASE ADMINISTRATOR

The effectiveness of a data base is largely dependent upon the degree of involvement of the DBA in coordinating and carrying out the design and maintenance of the data base. A typical role for the DBA is desribed in Bandurski and Jefferson.[4]

> "The data base administrator plays a key role in CAD [Computer Aided Design]. He must design and maintain the data structures used by a diversity of designers and applications programs, so must be familiar with the entire design process as well as data management. Ideally, he should provide interfaces to protect the user from unnecessary details of programming and data structures. Each user should view the data in as natural a form as possible, even though the actual physical structure as determined by the data base administrator, is quite different. It would be the task of the data base administrator, for example, to provide implicit links to satisfy infrequent queries, and highly efficient explicit links to provide data to the pre-programmed analysis routines."

The DBA must insure that the logical structure is designed to allow for its efficient use by applications programs. Specifically, he must be concerned with minimizing the operating cost of the data base and with eliminating unnecessary I/O and redundance in the structure while still accommodating different views of the data base. He must also try to make the interface between human and computer as smooth as possible.

The addition of GIRS to COMRADE will aid the DBA in handling these problems. Of course, the DBA will have to satisfy certain GIRS parameters. One of the main concerns of the DBA will be to determine how best to partition the graph onto GIRS pages and continuants. This involves determining

an optimum continuant size, the maximum number of continuants to be in main memory at any one time, and also which (classes of) relationships are to reside on which pages and continuants. Although determination of page and continuant residence can be left to default (onto a single page with successive continuants) fine-tuning the graph partition will improve performance. This is not a trivial task. To do this fine tuning, the DBA must design a STRATEGY. This is described in the section, "Removal of Pointers."

The DBA will perform other tasks also. He must determine what extra information is needed to answer an imprecise query (see section, "Satisfying an Imprecise Query") and whether or not the ability to answer it is worth the memory cost. It will also be up the DBA to assign a unique identification to every file associated with the data base, since the Cross File Reference Table must be unified across that data base.

The COMRADE/GIRS system offers speed as well as an option for greater flexibility at the cost of more storage, a trade-off which must be weighed by the DBA. The DBA would be charged with making the applications programmers fully aware of the potential of these new flexibilities.

POINTER OPERATIONS UNDER COMRADE AND UNDER COMRADE/GIRS

This section compares the methods of performing the major operations involved in creating and maintaining a data base under the COMRADE system and under COMRADE/GIRS. The following operations are considered:

- Defining data structures
- Loading pointers into the data base
- Updating and deleting data structure components
- Retrieving data from the data base
- Copying all or part of substructure

Under COMRADE, pointers are treated in much the same manner as alphanumeric, integer, real, and text data. Therefore, many of the operations described will hold true for other data types.

DEFINING DATA STRUCTURES

The data structure is presently defined by creating data-block formats called block type definitions. The following description is taken from Martin and Bell[15], page 16.

> "A block type definition can be defined by executing permanent file COMRDMDEFINEBLOCKTYPEABS or the DEFINE BLOCK TYPE option of the COMRADE Command Procedure BTMAINT.
>
> The user must first define a suitable data structure. The elements must be logically grouped and the type of data each is to contain must be determined. Names and the block type may also be changed but elements may not be reorganized, deleted or added."

---

[15] Martin, R. and C. Bell, "A Primer for the COMRADE Data Management System," NSRDC Report 4605 (Jan 1975).

Under COMRADE/GIRS , the data structure would no longer need to adhere to a rigid, possibly confining, format. The data structure for each relationship would be defined at the time the relationship was added to the graph and so it could be easily changed, as will be shown in a later section. If desired, it may be defined prior to loading via a STRATEGY routine called by the GIRS pointer insertion routine INSERT. Use of STRATEGY is discussed later in the section, "Conversion of a Data Base for the COMRADE/GIRS System."

LOADING POINTERS INTO THE DATA BASE

Under the COMRADE system, the Bulk Data Loader performs the initial loading of the data base. The Loader can also add new data blocks to an existing data base.

To add the relationships of Figure 14 to the data base, the format in Figure 15 would be used.

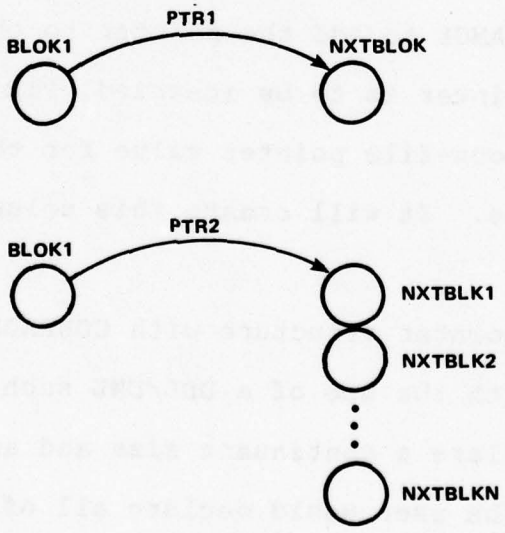


Figure 14 - BLOK1 - PTR Relationships

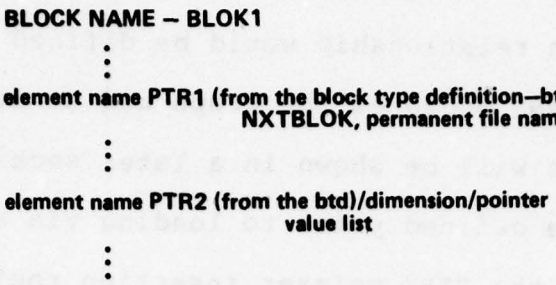


Figure 15 - Format of Input to Bulk Data Loader

Briefly, the Bulk Data Loader builds the data base by first creating a skeleton file for items such as a directory, sub-directory, inverted name list, reallocation table, using Subroutine DEFIL. Next, Subroutine DEFINB creates a skeleton data block. The interpreter (LOADB) then calls LDEL or LDRG, depending upon whether or not the element to be loaded is part of a repeating group. Either of these two routines will call on PTR and then CHANGE to add the pointer to the data base. If a cross-file pointer is to be inserted, PTR will call PFNIN. PFNIN returns a cross-file pointer value for the specified permanent file name. It will create this value if one does not already exist.

Creating the pointer structure with COMRADE/GIRS would best be accomplished with the use of a DDL/DML such as GIRL. First, the user would declare a continuant size and an initial number of pages. Next, the user would declare all of the block names (nodes) and pointer element names (links) in a DEFINE statement.

The nodes and links would be assigned unique sequence-dependent "random" numbers which the user would either store or regenerate for future update runs.

The user would then be ready to create the relationship graph. To insert the first relationship (referred to in Figures 14 and 15), in which the element name is PTR1, the following GIRL code would be required:

```
G    BLOK1  PTR1  NXTBLOK
```

FORTRAN code can be interleaved with GIRL code, allowing for attendant logic or calls to the CDMS subroutines. A multivalue list (pointer array), where the element name is PTR2, would be inserted similarly with the following code:

```
G    BLOK1  PTR2  (NXTBLK1, NXTBLK2,...)
```

The GIRL preprocessor would then convert these statements into FORTRAN calls to Subroutine INSERT. An entire FORTRAN program would then be created by the preprocessor and executed by the user. Let us say that the continuant size had been set to 500 words, and that BLOK1, PTR1, and NXTBLOK had been assigned the random numbers 326, 205, and 92, respectively. INSERT would then determine the necessary offset for placing 92 in the GIRS buffer by computing (326+205) MOD 500. It would then determine the appropriate page and continuant to receive this value by extracting a non-zero page and continuant number from BLOK1, or, if the page number had not yet been defined, by calling a user-defined STRATEGY for page definition. If the user defaulted, the triple would be placed on the page

and continuant last used. For further details, consult Berkowitz.[5]

UPDATING AND DELETING DATA STRUCTURE COMPONENTS

Under COMRADE, there are three methods of modifying and deleting (pointer) data. In the first method, a user calls the CDMS subroutines directly. In the other two methods, the subroutines are called either by the INTERACTIVEUPDATE procedure or by the Bulk Data Modifier. All these methods call on the following CDMS subroutines:

CHANGE	Adds or modifies non-repeating group elements of an existing data block.
CHANGO	Changes values in an occurrence of a repeating group in an existing data block.
ADDO	Adds a single occurrence of a repeating group in an existing data block.
ADDU	Creates a locally defined element to a data block.
DELETE	Deletes the value of an element in a specified block.

The following example illustrates the way in which a request to change a block name (sink node) is handled under COMRADE. Assume that the data block EQUIP1 contains a WEIGHT pointer to the data block BLOK05, and that we desire to have it point to BLOK07 instead. The parameters to CHANGE (ICODE, IBN, ISBN, IEN, DATA) are set as follows:

ICODE = 0 (does not have to be set for this example)

IBN = 6HEQUIP1

ISBN = 4HSUB1

IEN = 6HWEIGHT

DATA = 6HBLOK07

Under GIRL, this same request would look as follows:

```
G    EQUIP1  WEIGHT -.1  BLOK07
```

(Note that under COMRADE/GIRS, the parameter ISBN, which refers to the subblock number, is not needed for pointer updates.) The GIRL preprocessor would convert this code into a call to the GIRS subroutine INSERT.

This request to change a block name could also be handled by the routine PTREXEC, described later in the section which discusses the conversion of a data base to the COMRADE/GIRS system. Parameters for PTREXEC would be the same as for CHANGE, except that ISBN would be replaced by an operation code which, for this example, would be eight. The relationship in question can be deleted with the following GIRL code:

```
G    EQUIP1 - WEIGHT
```

Before a GIRS operation can be performed, all nodes and links must be defined -- that is, be given unique numeric representations. Any previously defined nodes and links must have their old values returned, and new nodes and links must be declared in a DEFINE statement. The user may store the old values on disk, or he may redeclare the old nodes and links following the same sequence as that used originally for the creation of the data structure.

RETRIEVING DATA FROM THE DATA BASE

Retrieval at a Higher Level

Data can be retrieved from a COMRADE data base at either of two different levels, a convenience which affords the user a flexibility versus convenience trade-off. The higher level offers more convenience for simple requests; the lower level offers fewer restrictions. At the higher level, the QUERY processor parses English-like requests from a remote terminal and converts them into CDMS subroutine calls. All possible hits are returned to the terminal. If the programmer wishes to make further use of the values, he must re-enter them into the machine. Moreover, he may not restrict a search to specific occurrences of repeating groups or to non-repeating group elements.

The queries are basically of three different types:

1. Query on condition.
2. Query via pointers.
3. Query on conditional values returned via pointers.

The examples provided earlier for each of the three types in the section entitled "Pointer Traversal Executive Routine" are repeated here for convenience.

```
1.  PRINT BN .WHERE. AREA .EQ. 20;
2.  PRINT COST .OF. BLOCK1/PTR1/PTR2;
3.  PRINT HEIGHT .OF. BLOCK2/PTR3/PTR4 .WHERE.
    AREA .GT. 1000;
```

A brief description of the COMRADE method of handling these queries has already been given. Under COMRADE/GIRS, operations for dealing with the first type of request would be the same.

For queries of the second type, the calls to FETCH to bring into main memory each and every data block involved in the search would no longer be made, since, under a COMRADE/GIRS system, a pointer chase could be processed largely in main memory via a single call to the pointer traversal executive routine PTRCHSE. PTRCHSE would convert an .OF. list from a query to a series of calls to the GIRS retrieval routine FIND. For example, the query

```
PRINT COST .OF. BLOCK1/PTR1/PTR2
```

would be handled as follows:

```
C     NODE = BLOCK1
      LINK = PTR1
      CALL FIND (NODE, LINK, VALUE)
      NODE = VALUE
      LINK = PTR2
      CALL FIND (NODE, LINK, VALUE)
C     VALUE NOW CONTAINS THE NAME OF THE HIT BLOCK
      IBN = VALUE
      CALL FETCH (...,IBN,...,DATA,...)
      PRINT DATA
```

If either PTR1 and PTR2 were pointer arrays, PTRCHSE would generate stacks to handle the situation. Also, as a worst case for the present system, if a pointer chase were to be expanded to its maximum of 16 levels,* resulting in a single hit block, and if the pointer relationships were partitioned properly in a GIRS graph, there would be as much as a 16 to 1 reduction in the use of disk I/O.

Another restriction under COMRADE * prevents a query from extending to more than one other file at a time. It does so

---

* This restriction holds only for the Query processor of COMRADE.

in a manner described in the section, "Cross-File Pointer Chasing in COMRADE." In contrast, COMRADE/GIRS would treat cross-file pointers in the same way as it treats intra-file pointers.

Under COMRADE, a type-3 query would require the QUERY processor to first perform a pointer chase (again bringing in all data blocks involved in the traversal) and to then test the specified values from the "hit" blocks against the conditions of the .WHERE. clause. Thus an inordinate amount of disk I/O would be used, both for pointer traversals and for those "hit" blocks which subsequently proved not to contain any values that satisfied the .WHERE. condition.

A COMRADE/GIRS system would avoid this excessive use of disk I/O in the following manner. All values to be conditionally tested, such as those returned by a type-3 query would be placed on the inverted list so that they could be treated as queries of the first type. Thus a type-3 query would result in two lists, one with the names of all the hit blocks from the pointer chase, and the other with the names of only those data blocks having values satisfying the .WHERE. condition. By ANDing these two lists, the names of only those blocks actually qualifying (to be brought in by FETCH) would be obtained.

Retrieval at a Lower Level

At the lower level, none of the restrictions associated with the QUERY language would apply. The user would set up

the necessary logic and then call the CDMS routines himself. Following are the major CDMS retrieval subroutines.

FETCH	Retrieves the value(s) of a single element, array, or repeating group.
FETCHR	Retrieves the values from a set of repeating group elements.
FETCHN	Retrieves the values from a set of non-repeating group elements.
QUERY	Returns the hits from a .WHERE. clause (this subroutine is not to be confused with the QUERY processor)

Suppose that a programmer wishes to determine the block name in that EQUIP block which is pointed out by the pointer WEIGHT, as shown in Figure 16.

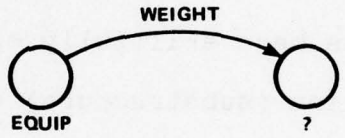


Figure 16 - EQUIP-WEIGHT Relationship

He might use the following code in calling FETCH.

```
IBN      =  5HEQUIP
ISBN     =  1
IRGN     =  0
IRGR     =  0
IEN      =  6HWEIGHT
LENGTH   =  1
CALL FETCH (IERR,IBN,ISBN,IRGN,IRGR,IEN,DATA,LENGTH)
```

FETCH would bring the EQUIP block into main memory and return a block name in the DATA parameter:

```
DATA = 6HBLOK20
```

If the programmer uses the PTREXEC facility, code such as the following might be used:

```
IBN   =  EQUIP
IEN   =  WEIGHT
IOP   =  5
CALL PTREXEC (IERR,IBN,IEN,IOP,DATA)
PRINT DATA
```

The GIRL code for this retrieval would be

```
G     EQUIP + WEIGHT' DATA
      PRINT DATA
```

COPYING/SEARCHING ALL OR A PART OF A SUBSTRUCTURE

As already mentioned, under the present COMRADE system, one and sometimes two disk accesses are required to bring a needed association into main memory. Consequently, any program which must copy all or any part of a substructure becomes seriously I/O bound. Under paged GIRS, on the other hand, and assuming that the structure has been skillfully enough partitioned so that the desired portion (substructure) of the graph is contained on a single continuant or page, the copy operation consists merely of indicating the appropriate page and continuant number of a user-callable GIRS report generator, LVDUMP. Each desired continuant is then transferred with a single FORTRAN READ followed by a WRITE.

If the user wishes to take advantage of Subroutine LVDUMP, but finds that the graph has not been properly partitioned beforehand to have the desired information isolated on a single page, additional steps must be performed. Two of the methods available for performing these steps are particularly suited for use with GIRS because they are essentially in-core operations and offer trade-offs as to time, flexibility, and

space. Both methods require an entry point to the graph (starting node). (Note that these two methods apply for searching through a graph, also.) The first method requires the user to supply a stack of pointers or link names at run time. Portions of this method form the basis for all three versions of Subroutine DELEQ, listed in Appendix C. The flow of operation of this first method is indicated in Figure 17. NODSTAK temporarily stores the values or sink nodes encountered.

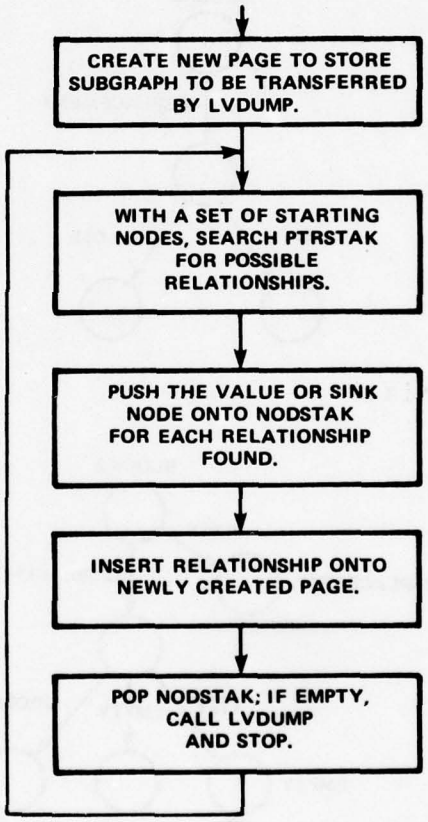
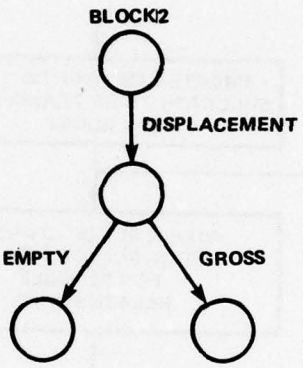


Figure 17 - Algorithm for Copying/Searching Tree-Structured Data

Under the second method, although it is more costly in terms of space, links need not be supplied at run time, since the programmer will already have supplied a list of links associated with each non-terminal node when the graph was created (Figure 18). All that is required for the copy or search operation to proceed is either a set of boundary (or terminal) blocks (nodes) or an integer value representing the depth or number of levels to be spanned.

This:



Becomes this:

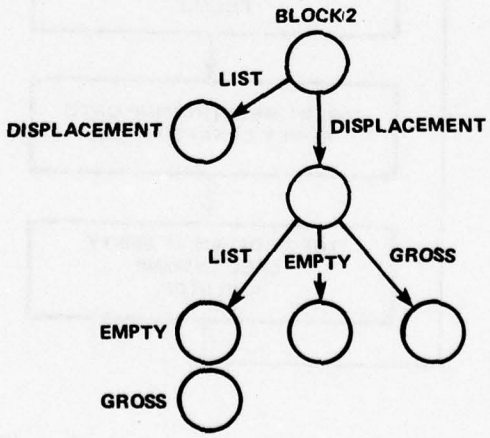


Figure 18 - Graph Modification for Copy/Search

In conclusion, copying (or searching) a graph structure would be easier, and would take less time to program and execute, under a COMRADE/GIRS system with GIRL than under the COMRADE system.

CONVERSION OF A COMRADE DATA BASE FOR THE COMRADE/GIRS SYSTEM

Conversion of a COMRADE data base to one suitable for use with GIRS would involve modification of the Block Type Definitions (BTD's), the Cross File Reference Tables, and the data base itself. All pointers would have to be eliminated from the data base and all pointer element names would have to be eliminated from the BTD's. Under the direction of the DBA, Cross File Reference Tables would have to be unified across a data base. Note, however, that present applications programs would not be affected, since COMRADE could be modified to call the GIRS routines directly when a pointer operation was recognized.

REMOVAL OF POINTERS

A program would have to be written (under the supervision of the DBA) to separate the pointers in the COMRADE data base from the data. Input to this program would consist of a complete list of all of the data blocks in the data base.

The DBA at this point, would define a continuant size, keeping in mind the time versus memory trade-off between long internal conflict lists and unused space in the GIRS continuant, as described by Berkowitz.[5] The program would then bring in each block and its block type definition. A call to the GIRS insertion routine (INSERT) would then be generated with the data block name as the source node, the pointer name from

the block type definition as the link name, and the value or new block name as the sink node of a GIRS pointer triple.

To partition the graph, the user would have the choice of either allowing GIRS to create a single page (letting the number of continuants grow as needed to handle graph overflow), or taking an active role in designing the replacement (or new) graph. This could be done by designing a STRATEGY routine.

STRATEGY would be called by INSERT and might contain the following instructions: "If the data block (source node) is an EQUIP block, place the triple on page two; if the pointer (link) is a WEIGHT pointer, place the triple on page three, continuant one; if the data block is DECK, create a back pointer to it from its value; otherwise, place the triple on the current or most recently accessed page."

When the pointer-removal procedure is complete, all of the pointers will have been removed from the COMRADE data base and placed into the GIRS buffer. More relationships can be added at this point with calls to PTREXEC, the pointer executive routine which is discussed in the next section. The data blocks and block type definitions would then be compressed and the cross file reference table would be updated, if necessary, resulting in a converted data base.

DEVELOPMENT OF NEW PROGRAMS INVOLVING POINTERS

The COMRADE/GIRS applications programmer would have a number of options for setting up a program with pointer operations:

(1) He could continue to call on the CDMS subroutines directly as at present, although this would be inefficient, since the CDMS routines must determine whether or not to call the GIRS routines.

(2) He could use GIRL, as imbedded in FORTRAN, which would save both programmer and computer time, and would also allow for a better conceptualization of the data base structure even though the use of GIRL would require a preprocessing step* already described in the section "Adding a Data-Definition/Data Manipulation Language to COMRADE."

(3) He could use PTREXEC, a pointer executive routine which would have a calling sequence similar to the present CDMS routines. Use of PTREXEC would avoid the inefficiencies mentioned in the first two methods but would not present a one-to-one trace of the pointer operation to the code.

PTREXEC would also translate Hollerith block names into their respective random numbers to be used by GIRS. The first input parameter to PTREXEC would be the operations code, an integer which would define the pointer operation. Table 5 contains a partial list of possible pointer operations.

---

* The GIRL preprocessor accepts a GIRL program as input and creates a new all FORTRAN applications program.

TABLE 5 - PTREXEC OPERATION CODES

Operation	Operation Number
Insert a new triple at the end of a list*	1
Insert a new triple in front of the n^{th} value in a list	2
Delete array (multivalue list)	3
Delete triple	4
Retrieve (first) value	5
Retrieve n^{th} value from the top of the list	6
Retrieve n^{th} value from the bottom of the list	7
Replace DATA value	8
* Note that if no list exists a new triple is created	

The other input parameters would be the block name for the source node (IBN), the pointer name for the link (IEN), and the value (DATA) as the sink node. ICODE is presently used for both input and output and would continue as such. For example, as an input parameter to the CDMS routine CHANGE, ICODE is used for arrays; under a COMRADE/GIRS system it would be used for multivalue lists. As an output parameter, it is an error code and its function would not change.

IMPLEMENTATION

Under COMRADE, pointers are treated as just another type of data, along with alphanumeric, integer, real, and text. Since under COMRADE/GIRS there would no longer be any pointer data within the data block, CDMS references to pointers would no longer be relevant and could be removed at the leisure of the COMRADE managers.

CDMS routines which deal specifically with pointers would have to be modified, replaced, or eliminated. The following areas of COMRADE would be involved:

In the QUERY program. To handle a parsed .OF. clause, subroutine PNTRCE (Overlay (3,0)) would be replaced by a subroutine (PTRCHSE) which would take the output of Subroutine PARSEP and treat the first word in the parsed list as a source node and the following words in the list as links and translate this list into one call per link or pointer to the GIRS retrieval subroutine FIND. PTRCHSE would then generate an array of hits or potential hits to be ANDed with the output list of a .WHERE. clause, thus reducing disk I/O. (The conditional elements involved in a query composed of both .WHERE. and .OF. clauses would then have to be inverted.) As a result, the number of hits per query would become available.

Overlay (3,1), which handles the data file switching to accommodate cross-file pointer chasing, would no longer be needed.

In the CDMS Subroutine Library. Under COMRADE, the cross file reference numbers are local to a particular file within a

COMRADE data base. This feature would be modified so that there would be a single Cross File Reference Table (CFRT), since pointers would no longer be oriented to particular data files. Each file in the data base would have a unique number associated with it. Therefore, this number must be packed into every sink node in the pointer graph. Data bases composed of only one file would not be affected. The following subroutines might be affected:

PTR. PTR is a routine in the Bulk Data Loader which sets the pointer word in main memory. It would no longer be needed.

INPFN. INPFN is a retrieval routine which returns the permanent file name from the Cross File Reference Table. The input to this routine is a cross-file pointer value which has been packed into the block name. The function of this routine would not change; however, the routine would not be needed until the search was complete, at which time it would be called once per query and would return a list of permanent file names when handed a list of hits.

To prevent existing applications programs from being affected by the addition of GIRS, CDMS routines would have to be modified to recognize requests involving pointer information. A pointer operation may be assumed if a search of the proper Block Type Definition results in failure. The CDMS routine would then have to call the proper GIRS routine directly.

Certain software would have to be added to COMRADE.

(1) The GIRS subroutines would have to be added to COMRADE's subroutine library.

(2) Two new permanent files would have to be created; one, the Interactive Pointer Manipulation Package, to handle pointer manipulation at the terminal, and the other, a Bulk Pointer Data Program to handle batch jobs involving bulk data description input. The Interactive Pointer Manipulation Package would allow for updating, inserting, deleting, and retrieving data block relationships at the terminal. It would consist of two parts. One part, the program PTREXEC, would be a pointer executive routine with both long and short tutorials. This program would convert tutorial responses directly into calls to the GIRS subroutines. A similar program is already in existence. The other part would be a collection of subroutines created to parse and handle such typical graph manipulation operations or requests as:

- Delete all references to Data Block (Node) x
- Delete Node x and its descendants
- Delete Node x and reconnect its ancestor to all of its descendants
- Add Node x to List y
- Retrieve all of the descendants to Node x

(Of course, some of these operations would require that the graph be adequately constructed with back pointers and also that the pointer (link) names be available, in a stack, perhaps.)

The Bulk Pointer Data Program for use primarily in bulk pointer operations for batch input would process whatever DDL, such as GIRL, were used. A GIRL language preprocessor has already been made available, which allows FORTRAN statements (such as calls to routines in the CDMS subroutine library) to be interspersed with GIRL statements thus making possible the performance of efficient operations of the form:

Delete data block x at the end of the pointer search.

The proposed method of converting a COMRADE data base to a COMRADE/GIRS data base best serves the interests of both the applications programmer and the DBA. As more needs are defined, the implementation may be extended.

SUMMARY

The advantages of removing the logical structure (pointers) from the data blocks and collecting them into a single area are several:

- Most of the disk I/O presently needed for accessing and modifying data block relationships can be eliminated--and thus response time for a COMRADE retrieval reduced--since a large collection of pointers can be brought into main memory at one time.
- Partitioning of pointer sets can be flexible (at the cost of keeping a certain amount of space always "available"), so that a programmer need no longer be constrained by the predetermined data block formats known as Block Type Definitions.
- The data base structure can be conveniently operated on by a pointer manipulation scheme, since pointers are concentrated in a single area.
- Using such a pointer manipulation scheme enables the introduction of a powerful data-definition/data manipulation language such as GIRL (Graph Information Retrieval Language) into the system, with the following benefits:
  (i) Programming and debugging time can be reduced by virtue of the one-to-one correspondence between the DDL/DML code and the pointer relationships represented.

(ii) Imprecise queries can be handled, albeit at the expense of memory and disk space.

(iii) The data base administrator is provided with such features as (a) automatic addition of back pointers, and (b) automatic connection of the parents and offspring of a data block to be removed.

- The logical structure need not be created at the same time that the data base is created.
- The recognition and reporting out of large file structures becomes feasible.
- Only finally qualifying "hit" blocks are brought into main memory when both a pointer search and a conditional test are involved.

An effective scheme for manipulating pointers, known as GIRS (Graph Information Retrieval System), has already been developed in-house. This system, written in FORTRAN and readily portable, allows a user-defined "STRATEGY." Although a price must be paid for the gain in speed, flexibility, and convenience it provides, the amount of extra memory and disk space involved is generally in direct proportion to the degree of convenience and flexibility to be gained.

GIRS offers a new tool for the data base administrator. The DBA will, of course, be responsible for (1) determining certain GIRS parameters to optimize performance, and (2) determining how best to partition the relationship graph. Fortunately, existing applications programs need not be con-

verted, since the pertinent COMRADE routines can be modified to call the GIRS routines.

In conclusion, the speed, flexibility, traceability (in the case of GIRL), and the manageability of the proposed COMRADE/GIRS system are the most compelling arguments for the implementation of such a system. The advantages must be weighed against the additional space requirements and the extra costs involved.

ACKNOWLEDGMENTS

The contributions and helpfulness of T. Rhodes and S. Willner, both of DTNSRDC, are gratefully acknowledged.

APPENDIX A

THE "PRESIDENTS" DATA BASE

The "Presidents" Data Base[8] has been designed by Stanley E. Willner of DTNSRDC for tutorial purposes.[1] It contains political and personal information on the U.S. presidents and is structured as illustrated in Figure 19.

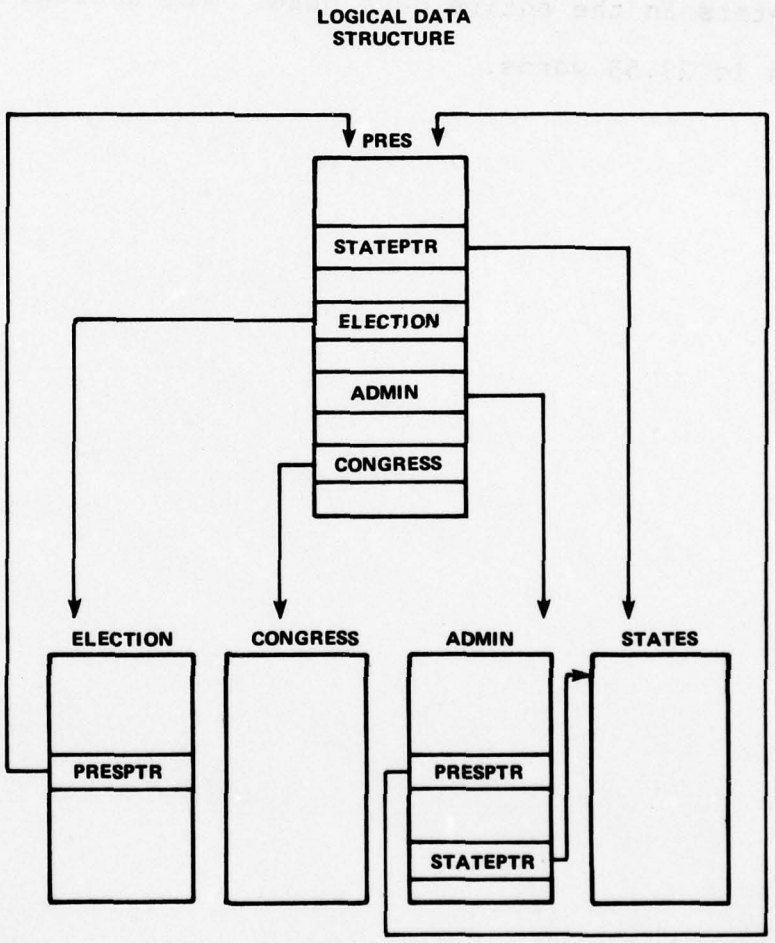


Figure 19 - Structure of "Presidents" Data Base

The "Presidents" Data Base is composed of block types (data block formats) PRES, ELECTION, CONGRESS, ADMIN, and STATES, as illustrated by Figures 20 through 24, respectively. There are 35 data blocks of the type PRES, one for each president; 45 data blocks of the type ELECTION, one for each presidential election up to 1964; 53 data blocks of the type ADMIN; 90 of the type CONGRESS; and 50 of the type STATES. Altogether there are 281 data blocks, two subdirectories, and 291 pointers in the entire data base. The average data block length is 23.55 words.

```
BLOCK TYPE-PRES

SUB-BLOCK  1- PERSONAL

ELEMENT    1- SURNAME    ALPHA              INVERTED
ELEMENT    2- FIRSTNAM   ALPHA
ELEMENT    3- INITIAL    ALPHA
ELEMENT    4- MONTHB     ALPHA
ELEMENT    5- DAYB       INTEGER
ELEMENT    6- YEARB      INTEGER            INVERTED
ELEMENT    7- STATEB     ALPHA              INVERTED
ELEMENT    8- STATEPTR   POINTER
ELEMENT    9- HEIGHT     ALPHA
ELEMENT   10- PARTY      ALPHA              INVERTED
ELEMENT   11- COLLEGE    ALPHA
ELEMENT   12- ANCESTRY   ALPHA              INVERTED
ELEMENT   13- RELIGION   ALPHA              INVERTED
ELEMENT   14- OCCUP      ALPHA       ARRAY
ELEMENT   15- MONTHD     ALPHA
ELEMENT   16- DAYD       INTEGER
ELEMENT   17- YEARD      INTEGER
ELEMENT   18- CAUSE      ALPHA

REPEATING GROUP  1- NAME
                    ELEMENT    1
                    ELEMENT    2
                    ELEMENT    3

REPEATING GROUP  2- BIRTH
                    ELEMENT    4
                    ELEMENT    5
                    ELEMENT    6
                    ELEMENT    7

REPEATING GROUP  3- DEATH
                    ELEMENT   15
                    ELEMENT   16
                    ELEMENT   17
                    ELEMENT   18

SUB-BLOCK  2- FAMILY

ELEMENT    1- FATHER     ALPHA
ELEMENT    2- MOTHER     ALPHA
ELEMENT    3- WIFE       ALPHA
ELEMENT    4- MONTHM     ALPHA
ELEMENT    5- DAYM       INTEGER
ELEMENT    6- YEARM      INTEGER
ELEMENT    7- CHILDREN   INTEGER

REPEATING GROUP  1- MARRIAGE
                    ELEMENT    3
                    ELEMENT    4
                    ELEMENT    5
                    ELEMENT    6
                    ELEMENT    7

SUB-BLOCK  3- HISTORY

ELEMENT    1- ELECTION   POINTER    ARRAY
ELEMENT    2- ADMIN      POINTER    ARRAY
ELEMENT    3- CONGRESS   POINTER    ARRAY
```

Figure 20 - Block Type PRES

```
BLOCK TYPE-ELECTION

SUB-BLOCK  1- SUB1

ELEMENT   1- YEAR      INTEGER          INVERTED
ELEMENT   2- WINNER    ALPHA            INVERTED
ELEMENT   3- PRESPTR   POINTER
ELEMENT   4- WPARTY    ALPHA            INVERTED
ELEMENT   5- VOTES     INTEGER
ELEMENT   6- OPPNAME   ALPHA
ELEMENT   7- OPPPARTY  ALPHA
ELEMENT   8- OPPVOTES  INTEGER

REPEATING GROUP  1- OPPONENT
                    ELEMENT   6
                    ELEMENT   7
                    ELEMENT   8
```

Figure 21 - Block Type ELECTION

```
BLOCK TYPE-STATES

SUB-BLOCK  1- SUB1

ELEMENT   1- STATE     ALPHA            INVERTED
ELEMENT   2- YEARA     INTEGER          INVERTED
ELEMENT   3- CAPITAL   ALPHA
ELEMENT   4- AREA      INTEGER          INVERTED
ELEMENT   5- AREARANK  INTEGER          INVERTED
ELEMENT   6- POPUL     INTEGER          INVERTED
ELEMENT   7- POPRANK   INTEGER          INVERTED
ELEMENT   8- ELECVOTE  INTEGER
ELEMENT   9- CITY      ALPHA            INVERTED
ELEMENT  10- CITYPOP   INTEGER          INVERTED

REPEATING GROUP  1- CITIES
                    ELEMENT   9
                    ELEMENT  10
```

Figure 22 - Block Type STATES

```
BLOCK TYPE-ADMIN

SUB-BLOCK  1- SUB1

ELEMENT    1- MONTHI     ALPHA
ELEMENT    2- DAYI       INTEGER
ELEMENT    3- YEARI      INTEGER          INVERTED
ELEMENT    4- VPFIRNAM   ALPHA
ELEMENT    5- VPSURNAM   ALPHA            INVERTED
ELEMENT    6- PRESPTR    POINTER
ELEMENT    7- SECSTATE   ALPHA
ELEMENT    8- SECWAR     ALPHA
ELEMENT    9- SECTRES    ALPHA
ELEMENT   10- ATTYGEN    ALPHA
ELEMENT   11- NEWSTATE   ALPHA            INVERTED
ELEMENT   12- STATEPTR   POINTER

REPEATING GROUP  1- INAUG
                    ELEMENT   1
                    ELEMENT   2
                    ELEMENT   3

REPEATING GROUP  2- VICEPRES
                    ELEMENT   4
                    ELEMENT   5

REPEATING GROUP  3- CABINET
                    ELEMENT   7
                    ELEMENT   8
                    ELEMENT   9
                    ELEMENT  10

REPEATING GROUP  4- STATES
                    ELEMENT  11
                    ELEMENT  12
```

Figure 23 - Block Type ADMIN

```
BLOCK TYPE-CONGRESS

SUB-BLOCK  1- SUB1

ELEMENT    1- NUMBER     INTEGER          INVERTED
ELEMENT    2- SPARTY     ALPHA
ELEMENT    3- SENATORS   INTEGER
ELEMENT    4- HPARTY     ALPHA
ELEMENT    5- REPS       INTEGER

REPEATING GROUP  1- SENATE
                    ELEMENT   2
                    ELEMENT   3

REPEATING GROUP  2- HOUSE
                    ELEMENT   4
                    ELEMENT   5
```

Figure 24 - Block Type CONGRESS

APPENDIX B

SEARCH PROCEDURE WRITTEN IN GIRL/FORTRAN FOR INDIRECT, MEMORY-RELATED QUERIES

The following procedure taken from the report by Berkowitz[5] indicates code typical of search procedures.

```
INFER:  B of A given that A is of type C.  If (C=0) RETURN

C    ***    LISTA HOLDS DOWN LINKS OF C
G     2     LISTA(HOLDS C 'TEMP, REFER TOP)
            JTEMP=0
      3     J=0
G     4     TEMP + DOWN/6 .'J=J+1'/6=B/'NEXT 5/7
G     5     LISTA(HOLDS NEXT, REFER 'JTEMP'//4)
G     6     LISTA(+HOLDS.'JTEMP=JTEMP+1' /15/'TEMP3)
C    ***    COMPILE STRING OF LINKS FROM C DOWN TO B
G     7     LISTA STRING(B,TEMP)
G     8     LISTA(+REFER.JTEMP'JTEMP=TOP//9,+HOLDS.JTEMP'TEMP,STRING
            TEMP//8)
C    ***    GENERATE LISTB:TREE OF NODES BASED ON LINK STRING
      9     J=0
G           A'NODE
G    10     LISTA+STRING.'J=J-1''TEMP=B//13
            K=0
G    11     NODE+TEMP.'K=K+1'/12'NEXT
G           LISTB(STRING NEXT,REFER 'J'//11)
G    12     LISTB(+STRING/16(.1'NODE,-.1,+REFER(.1'J,-.1//10)))
C    ***    OUTPUT NODES LINKED BY B
     13     JJ=0
G    14     NODE+B.'JJ=JJ+1'/12'OUT
            PRINT(OUT)
            GO TO 14
     15     PRINT('FAIL')
     16     RETURN
            END
```

APPENDIX C

TWO SUBROUTINES, AS CODED IN FORTRAN FOR CDMS AND GIRS, AND AS CODED IN GIRL

Two subroutines, DELEQ and DLEQPT, have been selected from the DTNSRDC Ship Design File to illustrate the coding differences for COMRADE and for COMRADE/GIRS. DELEQ is used to delete a particular data block and all of its descendants from the Ship Design File. DLEQPT is used to disconnect any pointers pointing to the deleted blocks and to reconnect them to the parent of the deleted block.

DELEQ operates on a simple "preorder" traversal technique for both CDMS and GIRL versions, whereas DLEQPT uses a much more involved algorithm which will not be considered here. The sequence of events for the GIRL version of DLEQPT is similar to that used in the CDMS version.

The GIRS version of both routines has been produced by the GIRL preprocessor. Handwritten GIRS code would be a bit more efficient and would also take less space in main memory. Table 6 compares the number of statements needed to accomplish the task under CDMS and under GIRL.

TABLE 6 - RATIO OF GIRL STATEMENTS TO COMRADE STATEMENTS

Subroutine Name	Number of Statements Needed		
	COMRADE	GIRL	Ratio of GIRL Statements to COMRADE Statements
DELEQ	107	42	.3925...
DLEQPT	160	71	.44375
Total	267	113	.4232...

One final comment: After becoming familiar with the two routines, the author was able to write the GIRL code for DELEQ in approximately half a day, and that for DLEQPT in approximately one and a half days.

CDMS, GIRL, and GIRS code follows:

AS CODED IN FORTRAN FOR CDMS

```
SUBROUTINE DELEQ      74/74   OPT=0 ROUND=*/ TRACE          FTN 4.5+R406

 1              SUBROUTINE DELEQ(EQIN)
               C
               C-THIS ROUTINE DELETES AN EQ BLOCK(EQIN) AND ALL SUBBLOCKS FROM THE
               C-SHIP DESIGN FILE, AND DELETES ALL POINTERS TO THOSE BLOCKS.
 5             C
                     COMMON/UNITS/LFN,ICONN
                     DIMENSION P(2),IPV(2),STACK(100),DATA(100),IP(100)
                     DATA P/2HAR,2HSW/,NPTRS/2/,IPV/1,2/
               C-INITIALIZE VARIABLES
10                   ISUM = 0
                     ISBN = 4HPTRS
                     PNOT1 = 8LNOTONE
                     SEVSIX = 767676767676767676768
                     ISTACK = 0
15                   PEQ = 2HEQ
               C-SET EQCUR TO EQIN
                     EQCUR = EQIN
               C-PLACE EQCUR ON STACK
                 10  ISTACK = ISTACK + 1
20                   STACK(ISTACK) = EQCUR
                     ISUM = 0
               C-FETCH THE POINTERS FROM EQCUR TO EACH OF THE POSSIBLE P BLOCKS, SEE
               C-WHICH IS EQUAL TO NOTONE, AND SET THE PROPER IP(ISTACK) TO THE SUM OF
               C-THE PROPER IPV'S.
25                   IRGR = 0
                     DO 30 I=1,NPTRS
                     EN = P(I)
                     LENGTH = 1
                     CALL COMRLOR(5LFETCH,IERR,EQCUR,ISBN,0,IRGR,EN,EOUT,LENGTH)
30                   IF(IERR.NE.0) CALL ERR(5HFETCH,IERR)
               C-SET IP SUMMING COUNTER ISUM
                     IF(EOUT.EQ.PNOT1) ISUM=ISUM+IPV(I)
                 30  CONTINUE
               C-OUT OF LOOP-SET IP(ISTACK)
35                   IP(ISTACK) = ISUM
               C-DOES EQCUR HAVE ANY SUB-EQ'S? FETCH THE DOWN POINTERS OF EQCUR
                 40  IRGR = 0
                     LENGTH = 100
                     CALL COMRLOR(5LFETCH,IERR,EQCUR,ISBN,5HDPTRS,IRGR,2HEQ,DATA,LENGTH
48                  1)
                     DO 45 JK=1,LENGTH
                     IF(DATA(JK).NE.SEVSIX) GO TO 46
                 45  CONTINUE
                     IF(IERR.NE.0) CALL ERR(5HFETCH,IERR)
45             C-ALL SEVEN SIXES INDICATE NO SUB-EQS
                     GO TO 50
               C-EQCUR DOES HAVE SUBEQ'S. SET EQCUR TO FIRST SUB-EQ, THE GO BACK AND
               C-PLACE IT ON STACK AND SET IP
                 46  EQCUR = DATA(JK)
50                   GO TO 10
               C-EQCUR DOES NOT HAVE SUB-EQ'S.  HOW MANY BLOCKS ARE IN STACK.
                 50  IF(ISTACK.EQ.1) GO TO 110
               C-SET IPDIF
                     IPDIF = IP(ISTACK-1)-IP(ISTACK)
55             C-FOLLOWING LOOP DELETES ALL BACK POINTERS TO EQCUR
                     DO 80 I=1,NPTRS
               C-SET ISUB
                     ISUB = 2**(NPTRS-I)
                     IF(IPDIF.LT.ISUB) GO TO 80
60             C-IPDIF IS GE ISUB.  FIRST SET PTR
                     PTR = P(NPTRS-I+1)
               C-FETCH VALUE OF PTR OF EQCUR AND PLACE IT IN PTCUR
                     IRGR = 0
                     LENGTH = 1
65                   CALL COMRLOR(5LFETCH,IERR,EQCUR,ISBN,0,IRGR,PTR,PTCUR,LENGTH)
                     IF(IERR.NE.0) CALL ERR(5HFETCH,IERR)
```

AD-A055 097 DAVID W TAYLOR NAVAL SHIP RESEARCH AND DEVELOPMENT CE--ETC F/G 9/2
FEASIBILITY STUDY FOR INCORPORATING A DATA STRUCTURE DEFINITION--ETC(U)
MAY 78 I S ZARITSKY
DTNSRDC-78/045

UNCLASSIFIED NL

2 OF 2
AD
A055097

END
DATE
FILMED
7-78
DDC

```
C-NOW MUST DELETE EQCUR FROM THE EQ FIRS OF PTCUR.  FIRST MUST FIND
C-WHICH OCCURRENCE OF EQ/PTCUR IS EQUAL TO EQCUR.  FETCH ALL EQ
C-OCCURRENCES
78          IRGR = 0
            LENGTH = 100
            IF(PTR.NE.2HSW) GO TO 55
            EQRG = 8HITEMPTRG
            IDPTR = 7HITEMPIR
75          GO TO 58
  55        EQRG = 4HEQRG
            IDPTR = 2HEQ
  58        CALL COMRLOR(5LFETCH,IERR,PTCUR,ISBN,EQRG,IRGR,IDPTR,DATA,LENGTH)
            IF(IERR.NE.0) CALL ERR(5HFETCH,IERR)
88          C-LOOP TO SEE WHICH OCCURRENCE IS EQCUR AND THEN DELETE IT.
            DO 60 K=1,LENGTH
            IF(EQCUR.NE.DATA(K)) GO TO 60
            C-EQCUR IS THE KTH OCCURRENCE IN EQ/PTCUR.  DELETE IT.
            CALL COMRLOR(6LDELETE,IERR,PTCUR,ISBN,EQRG,K,0)
85          IF(IERR.NE.0) CALL ERR(6HDELETE,IERR)
            GO TO 70
  60        CONTINUE
            C-RESET IPDIF-CONTINUE LOOPING IN MAJOR LOOP
  70        IPDIF = IPDIF-ISUB
90          80  CONTINUE
            C-DELETE BLOCK EQCUR FROM THE SHIP DESIGN FILE
            CALL COMRLOR(6LDELETB,IERR,EQCUR)
            C-THIS ROUTINE WILL BUILD BACK POINTERS FROM THAT OTHER BLOCK TO EACH
            IF(IERR.NE.0) CALL ERR(6HDELETB,IERR)
95          C-SET EQLST TO EQCUR
            EQLST = EQCUR
            C-SET EQCUR TO NEXT BLOCK ON STACK
            ISTACK = ISTACK-1
            EQCUR = STACK(ISTACK)
100         IRGR = 0
            LENGTH = 100
            CALL COMRLOR(5LFETCH,IERR,EQCUR,ISBN,5HDPTRS,IRGR,2HEQ,DATA,LENGTH
           1)
            C-DELETE EQLST FROM EQ/EQCUR(DOWN POINTER)  FIRST FETCH ALL DOWN
105         C-POINTERS EQ/EQCUR
            IF(IERR.NE.0) CALL ERR(5HFETCH,IERR)
            C-DO LOOP TO FIND EQLST AND DELETE IT.
            DO 90 K=1,LENGTH
            IF(EQLST.NE.DATA(K)) GO TO 90
110         C-DELETE IT AS IS KTH OCCURRENCE
            CALL COMRLOR(6LDELETE,IERR,EQCUR,ISBN,5HDPTRS,K,0)
            IF(IERR.NE.0) CALL ERR(6HDELETE,IERR)
            GO TO 100
  90        CONTINUE
115         C-START PROCESS AGAIN WITH NEW EQCUR
  100 GO TO 40
```

```
C-ONLY 1 BLOCK LEFT ON STACK.  FIRST SET EQPAR TO THE PARENT OF EQIN
  110 IRGR = 1
      LENGTH = 1
128   CALL COMRLOR(5LFETCH,IERR,EQIN,ISBN,5HBPTRS,IRGR,2HPR,EQPAR,LENGTH
     1)
      IF(IERR.NE.0) CALL ERR(5HFETCH,IERR)
C-FIND OUT IF EQPAR IS INDEED AN EQ BLOCK.  GET FIRST 2 CHARACTERS
C-OF EQPAR WITH MOVE AND TEST ON IT.
125   ARRAY1 = 10H
      CALL COMRLOR(4LMOVE,ARRAY1,1,EQPAR,1,2,IERR)
      IF(IERR.NE.0) CALL ERR(4HMOVE,IERR)
C-CHECK FIRST 2 CHARACTERS AGAINST EQ
      IF(ARRAY1.NE.PEQ) GO TO 140
130   C-EQPAR IS AN EQ BLOCK.  LOOP TO DELETE BACK POINTERS TO IT.
      DO 120 K=1,NPTRS
C-SET PTR TO P(K) TO GET TYPE OF POINTER.
      PTR = P(K)
C-PLACE VALUE OF PTR OF EQCUR IN VARIABLE PTCUR
135   IRGR = 0
      LENGTH = 1
      CALL COMRLOR(5LFETCH,IERR,EQCUR,ISBN,0,IRGR,PTR,PTCUR,LENGTH)
      IF(IERR.NE.0) CALL ERR(5HFETCH,IERR)
C-CALL DLEQPT TO DELETE TO AND FROM PTRS AND MAINTAIN HIERARCHY OF PTRS
148   CALL DLEQPT(EQCUR,PTR)
C-END OF LOOP
  120 CONTINUE
C-DELETE EQCUR FROM EQCS OF EQPAR.  FIRST FETCH ALL EQCS OF EQPAR
      IRGR = 0
145   LENGTH = 100
      CALL COMRLOR(5LFETCH,IERR,EQPAR,ISBN,5HDPTRS,IRGR,2HEQ,DATA,LENGTH
     1)
      IF(IERR.NE.0) CALL ERR(5HFETCH,IERR)
C-DO LOOP TO FIND EQCUR AND DELETE IT.
150   DO 130 K=1,LENGTH
      IF(EQCUR.NE.DATA(K)) GO TO 130
C-EQCUR IS KTH OCCURRENCE - DELETE IT
      CALL COMRLOR(6LDELETE,IERR,EQPAR,ISBN,5HDPTRS,K,0)
      IF(IERR.NE.0) CALL ERR(6HDELETE,IERR)
155   GO TO 140
  130 CONTINUE
C-DELETE EQCUR FROM SHIP DESIGN FILE
  140 CALL COMRLOR(6LDELET8,IERR,EQCUR)
      IF(IERR.NE.0) CALL ERR(6HDELET8,IERR)
160   C-FINISHED-RETURN
      RETURN
      END

STATISTICS
  PROGRAM LENGTH                14148      780
  CM LABELED COMMON LENGTH         28        2
```

```
 SUBROUTINE DLEQPT     74/74   OPT=0 ROUND=*/ TRACE            FTN 4.5+R406

 1               SUBROUTINE DLEQPT(EQIN,PTR)
                 C
                 C-GIVEN AN EQ BLOCK(EQIN) AND A POINTER TYPE(PTR) TO ANOTHER BLOCK,
                 C-THIS ROUTINE:
 5               C-       1- DELETES THE POINTER TO THE OTHER BLOCK
                 C-       2- DELETES THE BACK POINTER FROM THE OTHER BLOCK TO THE EQ
                 C-          BLOCK
                 C-       3- UPDATES POINTERS FROM AND BACK POINTERS TO HIGHER EQ
                 C-          BLOCKS AS NECESSARY TO MAINTAIN THE HIERARCHICAL SEQUENCE
 10              C-          OF (NOTONE(S, REAL POINTERS, AND (  LIMBOCS.
                 C
                       COMMON/UNITS/LFN,ICONN
                       DIMENSION DATA(100),DAT(100),ARRAY(2)
                 C-INITIALIZE VARIABLES
 15                    PEQ = 10HEQ
                       ISBN = 4HPTRS
                       PLIMBO = 6HLIMBO
                       FNOT1 = 8LNOTONE
                       IF(PTR.NE.2HSW) GO TO 3
 20                    EQRG = 8HITEMPTRG
                       ELEM = 7HITEMPTR
                       GO TO 4
                    3  EQRG = 4HEQRG
                       ELEM = 2HEQ
 25                 4  CONTINUE
                       ICODE = 0
                       IERR = 0
                 C-FETCH THE PTR OF EQIN AND PLACE ITS VALUE IN PTRINC
                       IRGR = 0
 30                    LENGTH = 1
                       CALL COMRLOR(5LFETCH,IERR,EQIN,ISBN,0,IRGR,PTR,PTRIN,LENGTH)
                       IF(IERR.NE.0) CALL ERR(5HFETCH,IERR)
                 C-SET PTCUR AND EQCUR TO EQIN
                       ARRAY1 = PTR
 35                    CALL COMRLOR(4LMOVE,ARRAY1,3,PLIMBO,1,5,IERR)
                       IF(IERR.NE.0) CALL ERR(4HMOVE,IERR)
                 C-IS PTRIN A LIMBO BLOCK
                       IMACH = ICAM(PTRIN,1,ARRAY1,1,7)
                       IF(IMACH.NE.0) GO TO 5
 40              C-PTRIN IS A LIMBO BLOCK.  BEFORE RETURNING DELETE EQCUR AS AN EQ/PTR
                 C-OF PTRIN.  FIRST, FETCH ALL EQ/PTRS OF PTRIN
                       LENGTH = 100
                       IRGR = 0
                       CALL COMRLOR(5LFETCH,IERR,PTRIN,ISBN,EQRG,IRGR,ELEM,DATA,LENGTH)
 45                    IF(IERR.NE.0) CALL ERR(5HFETCH,IERR)
                       DO 2 I=1,LENGTH
                       IF(EQIN.NE.DATA(I)) GO TO 2
                 C-DELETE EQCUR FROM EQ/PTRIN
                       CALL COMRLOR(6LDELETE,IERR,PTRIN,ISBN,EQRG,I,0)
 50                    IF(IERR.NE.0) CALL ERR(6HDELETE,IERR)
                       GO TO 180
                    2  CONTINUE
                       GO TO 180
                 C-SET PTCUR AND EQCUR
 55                 5  PTCUR = PTRIN
                       EQCUR = EQIN
                 C-DELETE PTR/POINTER OF EQCUR TO PTCUR
                   10  IRGR = 0
                       CALL COMRLOR(6LDELETE,IERR,EQCUR,ISBN,0,IRGR,PTR)
 60                    IF(IERR.NE.0) CALL ERR(6HDELETE,IERR)
                 C-SET EQLST TO EQCUR
                       EQLST = EQCUR
```

```
C-GET NEW EQCUR BY FETCHING PARENT OF CURRENT EQCUR AND PLACING ITS
C-VALUE IN EQCUR
65          IRGR = 1
            LENGTH = 1
            CALL COMRLDR(5LFETCH,IERR,EQCUR,ISBN,5HOPTRS,IRGR,2HPR,ESHOT,LENGT
           1H)
            IF(IERR.NE.0) CALL ERR(5HFETCH,IERR)
70          EQCUR = ESHOT
C-WHAT IS THE BLOCKTYPE OF EQCUR
            ARRAY1 = 10H
            CALL COMRLDR(4LMOVE,ARRAY1,1,EQCUR,1,2,IERR)
            IF(IERR.NE.0) CALL ERR(4HMOVE,IERR)
75          C-EQCUR IS NOT AN EQ BLOCK TYPE-DONE-RETURN
            IF(ARRAY1.NE.PEQ) GO TO 100
C-EQCUR IS AN EQ BLOCK TYPE.  SET PTLST TO PTCUR
            PTLST = PTCUR
C-SET A NEW PTCUR BY PLACING IN IT THE VALUE OF PTR OF EQCUR
80          LENGTH = 1
            IRGR = 0
            CALL COMRLDR(5LFETCH,IERR,EQCUR,ISBN,0,IRGR,PTR,PTCUR,LENGTH)
            IF(IERR.NE.0) CALL ERR(5HFETCH,IERR)
            IF(PTCUR.NE.PTRIN) GO TO 30
85          C-THE VALUE OF PTCUR IS PTRIN.  GET THE NUMBER OF DOWN POINTERS(EQ(S) OF
C-BLOCK EQCUR.
            IRGR = 0
            LENGTH = 100
            CALL COMRLDR(5LFETCH,IERR,EQCUR,ISBN,5HOPTRS,IRGR,2HEQ,DATA,LENGTH
90         1)
            IF(IERR.NE.0) CALL ERR(5HFETCH,IERR)
C-LENGTH IS 1, SO GO BACK ,SET EQCUR,ETC., AND GET NEXT ONE.
            IF(LENGTH.EQ.1) GO TO 10
C-LENGTH  GT  1, DONE--RETURN
95          IF(LENGTH.GT.1) GO TO 100
C-LENGTH IS  LT  1--ERROR--STOP
            WRITE(ICONN,20)
  20        FORMAT(1H ,30HERROR-EQ(S OF EQCUR MUST EXIT )
            CALL COMRLDR(4LFLFN,IERR,LFN)
100         IF(IERR.NE.0) CALL ERR(4HFLFN,IERR)
            STOP
C-THE VALUE OF PTRIN IS NOTONE.  SET IOC TO ZERO
  30  IOC = 0
C-IS PTLST EQUAL TO NOTONE
105         IF(PTLST.EQ.PNOT1) GO TO 60
C-PTLST IS NOT NOTONE.  DELETE EQLST FROM EQ PTRS OF PTLST.  FIRST MUST
C-FETCH ALL EQ(S FROM PTLST
            LENGTH = 100
            IRGR = 0
110         CALL COMRLDR(5LFETCH,IERR,PTLST,ISBN,EQRG,IRGR,ELEM,DATA,LENGTH)
            IF(IERR.NE.3) GO TO 60
C-PTLAT DOES NOT HAVE ANY EQ POINTERS OR EQLST IS NOT AMONG THOSE THAT
C-ARE THERE.  ERROR--ABORT
  40  WRITE(ICONN,50)
115   50  FORMAT(1H ,35HEQLST WAS NOT THERE TO DELETE-ABORT  )
            CALL COMRLDR(4LFLFN,IERR,LFN)
            IF(IERR.NE.0) CALL ERR(4HFLFN,IERR)
            STOP
C-NOW TRY TO DELETE EQLST
120   60  IF(IERR.NE.0) CALL ERR(5HFETCH,IERR)
            DO 70 I=1,LENGTH
            IF(EQLST.NE.DATA(I)) GO TO 70
            CALL COMRLDR(6LDELETE,IERR,PTLST,ISBN,EQRG,I,0)
            IF(IERR.NE.0) CALL ERR(6HDELETE,IERR)
125         GO TO 80
  70  CONTINUE
```

```
C-EQLST NOT AMONG EQCS OF EQCUR.  NEED TO KNOW HOW MANY THERE ARE(LEN)
  80  LEN = 100
      IRGR = 0
130   CALL COMRLDR(5LFETCH,IERR,EQCUR,ISBN,5HDPTRS,IRGR,2HEQ,DATA,LEN)
      IF(IERR.NE.0) CALL ERR(5HFETCH,IERR)
      IF(LEN.NE.1) GO TO 120
C-LEN IS ONE.  MUST DELETE EQLST FROM EQCS OF PTCUR
      IRGR = 0
135   LENGTH = 100
      CALL COMRLDR(5LFETCH,IERR,PTCUR,ISBN,EQRG,IRGR,ELEM,DATA,LENGTH)
      IF(IERR.NE.3) GO TO 100
C-PTCUR DOES NOT HAVE ANY EQ POINTERS OR EQLST IS NOT AMONG THOSE
C-EQCS THAT DO EXIST.  ERROR-ABORT
140   90  GO TO 40
C-NOW TRY TO DELETE EQLST
  100 IF(IERR.NE.0) CALL ERR(5HFETCH,IERR)
      DO 110 I=1,LENGTH
      IF(EQLST.NE.DATA(I)) GO TO 110
145   CALL COMRLDR(6LDELETE,IERR,PTCUR,ISBN,EQRG,I,0)
      IF(IERR.NE.0) CALL ERR(6HDELETE,IERR)
      GO TO 170
  110 CONTINUE
C-EQLST NOT AMONG EQCS OF PTCUR--ABORT
150   GO TO 90
C-LEN IS GREATER THAN ONE  NOW HAVE LOOP TO SEE ABOUT THE PTR OF EACH
C-SUBEQ OF EQCUR
  120 DO 140 I=1,LEN
C-SEE IF THE ITH SUBEQ IS EQLST.  IF IT IS CONTINUE
155   IF(DATA(I).EQ.EQLST) GO TO 140
C-CHECK ON IDO
      IF(IDO.NE.0) GO TO 130
C-IDC IS ZERO.  SET IT TO ONE AND SET SAME TO PTR OF DATA(I)
      IDO = 1
160   LENGTH = 1
      BN = DATA(I)
      IRGR = 0
      CALL COMRLDR(5LFETCH,IERR,BN,ISBN,0,IRGR,PTR,SAME,LENGTH)
      IF(IERR.NE.0) CALL ERR(5HFETCH,IERR)
165   GO TO 140
C-IDC WAS NOT ZERO.  SET PTRSUB TO PTR OF DATA(I)
  130 LENGTH = 1
      BN = DATA(I)
      IRGR = 0
170   CALL COMRLDR(5LFETCH,IERR,BN,ISBN,0,IRGR,PTR,PTRSUB,LENGTH)
      IF(IERR.NE.0) CALL ERR(5HFETCH,IERR)
C-IF PTRSUB IS NOT EQUAL TO SAME, DONE--RETURN
      IF(PTRSUB.NE.SAME) GO TO 180
C-END OF LOOP
175   140 CONTINUE
C-IF SAME IS NOTONE, DONE--RETURN
      IF(SAME.EQ.PNOT1) GO TO 180
```

```
C-SAME IS NOT NOTONE.  MUST DELETE EQ OF SAME POINTERS FROM ALL EQCS
C-UNDER PARENT(EQCUR).  FIRST, HAVE ALL EQCS OF EQCUR FROM ABOVE IN
C-ARRAY DATA.  NEXT, MUST FETCH ALL EQ PTRS OF SAME.
      LENGTH = 100
      IRGR = 0
      CALL COMRLOR(5LFETCH,IERR,SAME,ISBN,EQRG,IRGR,ELEM,DAT,LENGTH)
      IF(IERR.NE.3) GO TO 150
C-SAME HAS NO EQ PTRS, SO HAS NO SUBEQ PTRS
      GO TO 170
C-NOW GO THROUGH PROCESS OF DELETING FROM SAME ALL EQ PTRS WHICH ARE
C-SUBEQCS OF EQCUR
  150 DO 160 K=1,LEN
      DO 160 J=1,LENGTH
      IF(DATA(K).NE.DAT(J)) GO TO 160
      CALL COMRLOR(6LDELETE,IERR,SAME,ISBN,EQRG,J,99
      IF(IERR.NE.0) CALL ERR(6HDELETE,IERR)
  160 CONTINUE
C-ADD EQCUR AS AN EQ POINTER OF SAME
  170 IF(PTR.NE.2HSW) GO TO 175
      ARRAY(1) = 7HITEMPTR
      GO TO 176
  175 ARRAY(1) = 2HEQ
  176 ARRAY(2) = EQCUR
      IRGR = 0
      CALL COMRLOR(4LADDO,IERRA,SAME,ISBN,EQRG,IRGR,ARRAY,2)
      IF(IERRA.NE.0) CALL ERR(4HADDO,IERRA)
C-SET PTR OF EQCUR TO SAME
      CALL COMRLOR(6LCHANGE,ICODE,EQCUR,ISBN,PTR,SAME)
      IF(ICODE.NE.0) CALL ERR(6HCHANGE,ICODE)
C-SET EQLST TO EQCUR
      EQLST = EQCUR
C-SET VALUE OF EQCUR TO THE PARENT NAME OF THE PRESENT EQCUR
      LENGTH = 1
      IRGR = 1
      CALL COMRLOR(5LFETCH,IERR,EQCUR,ISBN,5HBPTRS,IRGR,2HPR,ESHOT,LENGT
     1H)
      IF(IERR.NE.0) CALL ERR(5HFETCH,IERR)
      EQCUR = ESHOT
C-FIND IF BLOCK TYPE OF EQCUR IS EQ OR NOT
      ARRAY1 = 10H
      CALL COMRLOR(4LMOVE,ARRAY1,1,EQCUR,1,2,IERR)
      IF(IERR.NE.0) CALL ERR(4HMOVE,IERR)
C-EQCUR IS NOT AN EQ BLOCK-DONE--RETURN
      IF(ARRAY1.NE.PEQ) GO TO 180
C-EQCUR IS AN EQ BLOCK.  SET PTCUR TO PTR OF EQCUR
      LENGTH = 1
      IRGR = 0
      CALL COMRLOR(5LFETCH,IERR,EQCUR,ISBN,0,IRGR,PTR,PTCUR,LENGTH)
      IF(IERR.NE.0) CALL ERR(5HFETCH,IERR)
C-GO TO WHERE GETTING THE SUBEQCS OF EQCUR AND RESUME PROCESS FROM
C-THERE.
      GO TO 80
C-END OF ROUTINE
  180 RETURN
      END
```

```
STATISTICS
  PROGRAM LENGTH                17058      965
  CM LABELED COMMON LENGTH         28        2
```

AS CODED IN GIRL


```
$     SUBROUTINE DELEQ(EQIN)
C
C-THIS ROUTINE DELETES AN EQ BLOCK(EQIN) AND ALL SUBBLOCKS FROM THE
C-SHIP DESIGN FILE, AND DELETES ALL POINTERS TO THOSE BLOCKS.
C
      INTEGER EQIN,HOLNAM,PEQ,PTR,ARRAY1,EQCUR,PTCUR,P,EQPAR,PARENT,
     + STACK
      COMMON /NAME/ LVNAME
      COMMON /NODES/ PNOT1,EQ,AR,SW,ITEMPTR,PR
      DIMENSION P(2),STACK(100)
G     DEFINE PNOT1,EQ,AR,SW,ITEMPTR,PR
      DATA NPTRS/2/
G     EXECUTE
C-INITIALIZE VARIABLES
      P(1)=AR
      P(2)=SW
      PEQ=2LEQ
      ISTACK = 0
C-SET EQCUR TO EQIN
      EQCUR = EQIN
C-PLACE EQCUR ON STACK
  10  ISTACK = ISTACK + 1
      STACK(ISTACK) = EQCUR
C    PERFORM PREORDER TRAVERSAL TO TERMINAL NODES
C-FETCH THE POINTERS FROM EQCUR TO EACH OF THE POSSIBLE P BLOCKS, SEE
C-WHICH IS EQUAL TO NOTONE, AND SET THE PROPER IP(ISTACK) TO THE SUM OF
C-THE PROPER IPV'S.
C-DOES EQCUR HAVE ANY SUB-EQ'S? FETCH THE DOWN POINTERS OF EQCUR
G 40  EQCUR+EQ/50 'EQCUR/10
C-EQCUR DOES NOT HAVE SUB-EQ'S.  HOW MANY BLOCKS ARE IN STACK.
  50  IF(ISTACK.EQ.1) GO TO 110
C-FOLLOWING LOOP DELETES ALL BACK POINTERS TO EQCUR
      DO 80 I=1,NPTRS
      PTR=P(I)
C-FETCH VALUE OF PTR OF EQCUR AND PLACE IT IN PTCUR
G     EQCUR+PTR/99 'PTCUR
      IDPTR=EQ
      IF(PTR.EQ.SW) IDPTR=ITEMPTR
C-NOW MUST DELETE EQCUR FROM THE EQ PTRS OF PTCUR.  FIRST MUST FIND
C-WHICH OCCURRENCE OF EQ/PTCUR IS EQUAL TO EQCUR.  FETCH ALL EQ
C-OCCURRENCES
C-LOOP TO SEE WHICH OCCURRENCE IS EQCUR AND THEN DELETE IT.
C-EQCUR IS THE KTH OCCURRENCE IN EQ/PTCUR.  DELETE IT.
      K=0
G 60  PTCUR+IDPTR/99(."K=K+1" =EQCUR/60,-.K)
  80  CONTINUE
C    DELETE TERMINAL BLOCK EQCUR FROM SHIP DESIGN FILE
G     EQCUR+LVNAME'HOLNAM
      CALL COMRLDR(6LDELETB,IERR,HOLNAM)
      IF(IERR.NE.0) CALL ERR(6HDELETB,IERR)
C-SET EQCUR TO NEXT BLOCK ON STACK
      ISTACK = ISTACK-1
      EQCUR = STACK(ISTACK)
C-DELETE EQLST FROM EQ/EQCUR(DOWN POINTER)  FIRST FETCH ALL DOWN
C-POINTERS EQ/EQCUR
G     EQCUR+EQ-.1
      GO TO 40
C-ONLY 1 BLOCK LEFT ON STACK.  FIRST SET EQPAR TO THE PARENT OF EQIN
C    IS PARENT AN EQ BLOCK
G 110 EQIN+PR/99 'PARENT+LVNAME'EQPAR
      ARRAY1= EQPAR.AND.7777000000000000000B
      IF(ARRAY1.NE.PEQ) GO TO 140
C-EQPAR IS AN EQ BLOCK.  LOOP TO DELETE BACK POINTERS TO IT.
C    DELETE POINTER FROM PARENT TO EQIN
      I=0
G 130 PARENT+EQ(."I=I+1"/99 =EQCUR/130,-.I)
G 140 EQCUR+LVNAME'HOLNAM
      CALL COMRLDR(6LDELETB,IERR,HOLNAM)
C-FINISHED-RETURN
      RETURN
C    DESIRED POINTER NOT FOUND - ERROR
  99  CALL ERR(5HFETCH,IERR)
G     COMPLETE
```

```
$     SUBROUTINE DLEQPT(EQIN,PTR)
C
C-GIVEN AN EQ BLOCK(EQIN) AND A POINTER TYPE(PTR) TO ANOTHER BLOCK,
C-THIS ROUTINE:
C-       1- DELETES THE POINTER TO THE OTHER BLOCK
C-       2- DELETES THE BACK POINTER FROM THE OTHER BLOCK TO THE EQ
C-          BLOCK
C-       3- UPDATES POINTERS FROM AND BACK POINTERS TO HIGHER EQ
C-          BLOCKS AS NECESSARY TO MAINTAIN THE HIERARCHICAL SEQUENCE
C-          OF (NOTONE'S, REAL POINTERS, AND (  LIMBO'S.
C
      INTEGER EQIN,HOLNAM,PEQ,PTR,ARRAY1,EQCUR,PTCUR,AR,DATA,EQ,PNOT1,
     + PTRNAM,SAME,TEMPNAM,ARRAY,BN,ELEM,EQLST,PLIMBO,PR,PTLST,PTRIN,
     + PTRSUB,SW
      COMMON /NAME/ LVNAME
      COMMON/UNITS/LFN,ICONN
      COMMON /NODES/ PNOT1,EQ,AR,SW,ITEMPTR,PR
C
G     EXECUTE
C-INITIALIZE VARIABLES
      PEQ=2LEQ
      PLIMBO = 6HLIMBO
      ELEM=EQ
      IF(PTR.EQ.SW) ELEM=ITEMPTR
C-FETCH THE PTR OF EQIN AND PLACE ITS VALUE IN PTRIN
G     EQIN+PTR/99 'PTRIN+LVNAME'PTRNAM
C-IS PTRIN A LIMBO BLOCK
      TEMPNAM=SHIFT(PTRNAM,12)
      IF(TEMPNAM.NE.PLIMBO) GO TO 5
C-PTRIN IS A LIMBO BLOCK.  BEFORE RETURNING DELETE EQCUR AS AN EQ/PTR
C-OF PTRIN.  FIRST, FETCH ALL EQ/PTRS OF PTRIN
C-DELETE EQCUR FROM EQ/PTRIN
      I=0
G 2   PTRIN+ELEM/99 (."I=I+1" =EQIN/2,-.I)
      RETURN
C-SET PTCUR AND EQCUR
  5   PTCUR = PTRIN
      EQCUR = EQIN
C-DELETE PTR/POINTER OF EQCUR TO PTCUR
C-SET EQLST TO EQCUR
C-GET NEW EQCUR BY FETCHING PARENT OF CURRENT EQCUR AND PLACING ITS
C-VALUE IN EQCUR
G 10  EQCUR'EQLST (-PTR,+PR'EQCUR+LVNAME'HOLNAM)
C-WHAT IS THE BLOCKTYPE OF EQCUR
      ARRAY1=HOLNAM.AND.777700000000000000008
C-EQCUR IS NOT AN EQ BLOCK TYPE-DONE-RETURN
      IF(ARRAY1.NE.PEQ) RETURN
C-EQCUR IS AN EQ BLOCK TYPE.  SET PTLST TO PTCUR
      PTLST = PTCUR
C-SET A NEW PTCUR BY PLACING IN IT THE VALUE OF PTR OF EQCUR
G     EQCUR+PTR/99 'PTCUR=PTRIN/30
C-THE VALUE OF PTCUR IS PTRIN.  GET THE NUMBER OF DOWN POINTERS(EQ'S) OF
C-BLOCK EQCUR.
C-LENGTH IS 1, SO GO BACK ,SET EQCUR,ETC., AND GET NEXT ONE.
C-LENGTH  GT  1, DONE--RETURN
C-LENGTH IS  LT  1--ERROR--STOP
G     EQCUR+EQ/98 .2/10/RETURN
 98   WRITE(ICONN,20)
 20   FORMAT(1H ,30HERROR-EQ'S OF EQCUR MUST EXIT )
      CALL COMRLDR(4LFLFN,IERR,LFN)
      IF(IERR.NE.0) CALL ERR(4HFLFN,IERR)
      STOP
C-THE VALUE OF PTRIN IS NOTONE.  SET IDO TO ZERO
 30   IDO = 0
C-IS PTLST EQUAL TO NOTONE
      IF(PTLST.EQ.PNOT1) GO TO 80
C-PTLST IS NOT NOTONE.  DELETE EQLST FROM EQ PTRS OF PTLST.  FIRST MUST
C-FETCH ALL EQ(S FROM PTLST
C-NOW TRY TO DELETE EQLST
      I=0
G 70  PTLST+ELEM/40 (."I=I+1"=EQLST/70,-.I)
      GO TO 80
```

```
C-PTLAT DOES NOT HAVE ANY EQ POINTERS OR EQLST IS NOT AMONG THOSE THAT
C-ARE THERE.  ERROR--ABORT
  40  WRITE(ICONN,50)
  50  FORMAT(1H ,35HEQLST WAS NOT THERE TO DELETE-ABORT  )
      CALL COMRLDR(4LFLFN,IERR,LFN)
      IF(IERR.NE.0) CALL ERR(4HFLFN,IERR)
      STOP
C-EQLST NOT AMONG EQ'S OF EQCUR.  NEED TO KNOW HOW MANY THERE ARE(LEN)
G 80  EQCUR+EQ/99 .2/85/120
C-LEN IS ONE.  MUST DELETE EQLST FROM EQ'S OF PTCUR
C- IF PTCUR DOES NOT HAVE ANY EQ POINTERS OR EQLST IS NOT AMONG THOSE
C-EQ'S THAT DO EXIST.  ERROR-ABORT
 85   I=0
G 110 PTCUR+ELEM/40(."I=I+1" /40 =EQLST/110,-.I)
      GO TO 170
C-LEN IS GREATER THAN ONE  NOW HAVE LOOP TO SEE ABOUT THE PTR OF EACH
C-SUBEQ OF EQCUR
C-SEE IF THE ITH SUBEQ IS EQLST.  IF IT IS CONTINUE
 120  I=0
G 121 EQCUR+EQ/99 ."I=I+1"/140 =EQLST'BN//121
C-CHECK ON IDO
      IF(IDO.NE.0) GO TO 130
C-IDO IS ZERO.  SET IT TO ONE AND SET SAME TO PTR OF DATA(I)
      IDO = 1
G     BN+PTR'SAME
      GO TO 140
C-IDO WAS NOT ZERO.  SET PTRSUB TO PTR OF DATA(I)
C-IF PTRSUB IS NOT EQUAL TO SAME, DONE--RETURN
G 130 BN+PTR/99'PTRSUB=SAME/RETURN/121
C-IF SAME IS NOTONE, DONE--RETURN
 140  IF(SAME.EQ.PNOT1) RETURN
C-SAME IS NOT NOTONE.  MUST DELETE EQ OF SAME POINTERS FROM ALL EQ(S
C-UNDER PARENT(EQCUR).  FIRST, HAVE ALL EQ(S OF EQCUR FROM ABOVE IN
C-ARRAY DATA.  NEXT, MUST FETCH ALL EQ PTRS OF SAME.
      I=0
G 150 EQCUR+EQ/99 ."I=I+1"/160 'DATA
C-NOW GO THROUGH PROCESS OF DELETING FROM SAME ALL EQ PTRS WHICH ARE
C-SUBEQ'S OF EQCUR
  160 J=0
G 155 SAME+ELEM/170(."J=J+1"/150=DATA/150,-.J)
      GO TO 150
C-ADD EQCUR AS AN EQ POINTER OF SAME
 170  IF(PTR.NE.SW) GO TO 175
      ARRAY=ITEMPTR
      GO TO 176
  175 ARRAY=EQ
G 176 SAME ARRAY EQCUR
C-SET PTR OF EQCUR TO SAME
G     EQCUR PTR -.1 SAME
C-SET EQLST TO EQCUR
      EQLST = EQCUR
C-SET VALUE OF EQCUR TO THE PARENT NAME OF THE PRESENT EQCUR
G     EQCUR+PR/99'EQCUR+LVNAME'HOLNAM
C-EQCUR IS NOT AN EQ BLOCK-DONE--RETURN
      ARRAY1=HOLNAM.AND.777700000000000000B
       IF(ARRAY1.NE.PEQ) RETURN
C-EQCUR IS AN EQ BLOCK.  SET PTCUR TO PTR OF EQCUR
G     EQCUR+PTR/99'PTCUR
C-GO TO WHERE GETTING THE SUBEQ'S OF EQCUR AND RESUME PROCESS FROM
C-THERE.
      GO TO 88
C     DESIRED POINTER NOT FOUND - ERROR
  99  CALL ERR(5HFETCH,IERR)
G     COMPLETE
/     COMPLETE
```

AS CODED IN FORTRAN FOR GIRS


```
SUBROUTINE DELEQ      74/74   OPT=0 ROUND=*/ TRACE          FTN 4.5+414

      SUBROUTINE DELEQ(EQIN)
      COMMON /LVARGS/ LVFUNC,LVVARG,LVVPOS,LVVTYP,LVVAL,
     +LVVNVL,LVSKIP,LVVTR,LVVINC,LVVALS(18),LVTYPE(18)
C
C-THIS ROUTINE DELETES AN EQ BLOCK(EQIN) AND ALL SUBBLOCKS FROM THE
C-SHIP DESIGN FILE, AND DELETES ALL POINTERS TO THOSE BLOCKS.
C
      INTEGER EQIN,HOLNAM,PEQ,PTR,ARRAY1,EQCUR,PTCUR,P,EQPAR,PARENT,
     + STACK
      COMMON /NAME/ LVNAME
      COMMON /NODES/ PNOT1,EQ,AR,SW,ITEMPTR,PR
      DIMENSION P(2),STACK(100)
      INTEGER
     +PNOT1,EQ,AR,SW,ITEMPTR,PR
      DATA NPTRS/2/
      CALL LVGRN(PNOT1 )
      CALL LVGRN(EQ    )
      CALL LVGRN(AR    )
      CALL LVGRN(SW    )
      CALL LVGRN(ITEMPT)
      CALL LVGRN(PR    )
      GO TO 25001
25000 CONTINUE
C-INITIALIZE VARIABLES
      P(1)=AR
      P(2)=SW
      PEQ=2LEQ
      ISTACK = 0
C-SET EQCUR TO EQIN
      EQCUR = EQIN
C-PLACE EQCUR ON STACK
   10 ISTACK = ISTACK + 1
      STACK(ISTACK) = EQCUR
C     PERFORM PREORDER TRAVERSAL TO TERMINAL NODES
C-FETCH THE POINTERS FROM EQCUR TO EACH OF THE POSSIBLE P BLOCKS, SEE
C-WHICH IS EQUAL TO NOTONE, AND SET THE PROPER IP(ISTACK) TO THE SUM OF
C-THE PROPER IPV'S.
C-DOES EQCUR HAVE ANY SUB-EQ'S? FETCH THE DOWN POINTERS OF EQCUR
   40 CONTINUE
C40   EQCUR+EQ/50 'EQCUR/10
      LVVAL=EQCUR
      LVVARG=LVVAL
      LVFUNC=EQ
      CALL LVFIND
      IF(LVVTR .EQ. -1) GO TO     50
      EQCUR=LVVAL
      IF(LVVTR .NE. -1) GO TO     10
C-EQCUR DOES NOT HAVE SUB-EQ'S.  HOW MANY BLOCKS ARE IN STACK.
   50 IF(ISTACK.EQ.1) GO TO 110
C-FOLLOWING LOOP DELETES ALL BACK POINTERS TO EQCUR
      DO 80 I=1,NPTRS
      PTR=P(I)
C-FETCH VALUE OF PTR OF EQCUR AND PLACE IT IN PTCUR
C     EQCUR+PTR/99 'PTCUR
      LVVAL=EQCUR
      LVVARG=LVVAL
      LVFUNC=PTR
      CALL LVFIND
      IF(LVVTR .EQ. -1) GO TO     99
      PTCUR=LVVAL
      IDPTR=EQ
      IF(PTR.EQ.SW) IDPTR=ITEMPTR
```

```
      C-NOW MUST DELETE EQCUR FROM THE EQ PTRS OF PTCUR.  FIRST MUST FIND
      C-WHICH OCCURRENCE OF EQ/PTCUR IS EQUAL TO EQCUR.  FETCH ALL EQ
65    C-OCCURRENCES
      C-LOOP TO SEE WHICH OCCURRENCE IS EQCUR AND THEN DELETE IT.
      C-EQCUR IS THE KTH OCCURRENCE IN EQ/PTCUR.  DELETE IT.
            K=0
       60   CONTINUE
70    C60   PTCUR=IDPTR/99(."K=K+1" =EQCUR/60,-.K)
            LVVAL=PTCUR
            LVVARG=LVVAL
            LVFUNC=IDPTR
            CALL LVFIND
75          IF(LVVTR .EQ. -1) GO TO     99
            LVV 1=LVVAL
            LVV 2=LVFUNC
            LVV 3=LVVARG
            K=K+1
80          LVVPOS=   K
            CALL LVFNV(LV  1)
            LVVTR=-1
            IF(LVVAL .EQ. EQCUR
           +) LVVTR=1
85          IF(LVVTR .EQ. -1) GO TO     60
            LVFUNC=LVV 2
            LVVARG=LVV 3
            CALL LVFIND
            LVVPOS=   K
90          CALL LVFNV(LV  2)
            CALL LVDLTI
         80 CONTINUE
      C     DELETE TERMINAL BLOCK EQCUR FROM SHIP DESIGN FILE
      C     EQCUR+LVNAME*HOLNAM
95          LVVAL=EQCUR
            LVVARG=LVVAL
            LVFUNC=LVNAME
            CALL LVFIND
            HOLNAM=LVVAL
100         CALL COMRLDR(6LDELETB,IERR,HOLNAM)
            IF(IERR.NE.0) CALL ERR(6HDELETB,IERR)
      C-SET EQCUR TO NEXT BLOCK ON STACK
            ISTACK = ISTACK-1
            EQCUR = STACK(ISTACK)
105   C-DELETE EQLST FROM EQ/EQCUR(DOWN POINTER)  FIRST FETCH ALL DOWN
      C-POINTERS EQ/EQCUR
      C     EQCUR+EQ-.1
            LVVAL=EQCUR
            LVVARG=LVVAL
110         LVFUNC=EQ
            CALL LVFIND
            CALL LVFNV(LV  3)
            CALL LVDLTI
            GO TO 40
115   C-ONLY 1 BLOCK LEFT ON STACK.  FIRST SET EQPAR TO THE PARENT OF EQIN
      C     IS PARENT AN EQ BLOCK
       110  CONTINUE
      C110  EQIN+PR/99 'PARENT+LVNAME'EQPAR
            LVVAL=EQIN
120         LVVARG=LVVAL
            LVFUNC=PR
            CALL LVFIND
            IF(LVVTR .EQ. -1) GO TO     99
            PARENT=LVVAL
125         LVVARG=LVVAL
            LVFUNC=LVNAME
            CALL LVFIND
            EQPAR=LVVAL
            ARRAY1= EQPAR.AND.7777000000000000000B
130         IF(ARRAY1.NE.PEQ) GO TO 140
```

```
C-EQPAR IS AN EQ BLOCK.  LOOP TO DELETE BACK POINTERS TO IT.
C     DELETE POINTER FROM PARENT TO EQIN
      I=0
  130 CONTINUE
C130  PARENT+EQ(."I=I+1"/99 =EQCUR/130,-.I)
      LVVAL=PARENT
      LVVARG=LVVAL
      LVFUNC=EQ
      CALL LVFIND
      LVV 1=LVVAL
      LVV 2=LVFUNC
      LVV 3=LVVARG
      I=I+1
      LVVPOS=   I
      CALL LVFNV(LV  4)
      IF(LVVTR .EQ. -1) GO TO     99
      LVVARG=LVVAL
      LVVTR=-1
      IF(LVVAL .EQ. EQCUR
     +) LVVTR=1
      IF(LVVTR .EQ. -1) GO TO    130
      LVFUNC=LVV 2
      LVVARG=LVV 3
      CALL LVFIND
      LVVPOS=   I
      CALL LVFNV(LV  5)
      CALL LVDLTI
  140 CONTINUE
C140  EQCUR+LVNAME'HOLNAM
      LVVAL=EQCUR
      LVVARG=LVVAL
      LVFUNC=LVNAME
      CALL LVFIND
      HOLNAM=LVVAL
      CALL COMRLDR(6LDELETB,IERR,HOLNAM)
C-FINISHED-RETURN
      RETURN
C     DESIRED POINTER NOT FOUND - ERROR
   99 CALL ERR(5HFETCH,IERR)
      RETURN
25001 CONTINUE
      LV 1=0
      LV 2=0
      LV 3=0
      LV 4=0
      LV 5=0
      GO TO 25000
      END
```

```
STATISTICS
  PROGRAM LENGTH                5478      359
  CM LABELED COMMON LENGTH       448       36
```

```
 SUBROUTINE DLEQPT     74/74   OPT=0 ROUND=*/ TRACE          FTN 4.5+414

 1               SUBROUTINE DLEQPT(EQIN,PTR)
                 COMMON /LVARGS/ LVFUNC,LVVARG,LVVPOS,LVVTYP,LVVAL,
                +LVVNVL,LVSKIP,LVVTR,LVVINC,LVVALS(10),LVTYPE(10)
                C
 5               C-GIVEN AN EQ BLOCK(EQIN) AND A POINTER TYPE(PTR) TO ANOTHER BLOCK,
                C-THIS ROUTINE:
                C-       1- DELETES THE POINTER TO THE OTHER BLOCK
                C-       2- DELETES THE BACK POINTER FROM THE OTHER BLOCK TO THE EQ
                C-          BLOCK
10              C-       3- UPDATES POINTERS FROM AND BACK POINTERS TO HIGHER EQ
                C-          BLOCKS AS NECESSARY TO MAINTAIN THE HIERARCHICAL SEQUENCE
                C-          OF (NOTONE'S, REAL POINTERS, AND (  LIMBO'S.
                C
                      INTEGER EQIN,HOLNAM,PEQ,PTR,ARRAY1,EQCUR,PTCUR,AR,DATA,EQ,PNOT1,
15                   + PTRNAM,SAME,TEMPNAM,ARRAY,BN,ELEM,EQLST,PLIMBO,PR,PTLST,PTRIN,
                     + PTRSUB,SW
                      COMMON /NAME/ LVNAME
                      COMMON/UNITS/LFN,ICONW
                      COMMON /NODES/ PNOT1,EQ,AR,SW,ITEMPTR,PR
20              C
                      GO TO 25001
                25000 CONTINUE
                C-INITIALIZE VARIABLES
                      PEQ=2LEQ
25                    PLIMBO = 6HLIMBO
                      ELEM=EQ
                      IF(PTR.EQ.SW) ELEM=ITEMPTR
                C-FETCH THE PTR OF EQIN AND PLACE ITS VALUE IN PTRIN
                C     EQIN+PTR/99 'PTRIN+LVNAME'PTRNAM
30                    LVVAL=EQIN
                      LVVARG=LVVAL
                      LVFUNC=PTR
                      CALL LVFIND
                      IF(LVVTR .EQ. -1) GO TO     99
35                    PTRIN=LVVAL
                      LVVARG=LVVAL
                      LVFUNC=LVNAME
                      CALL LVFIND
                      PTRNAM=LVVAL
40              C-IS PTRIN A LIMBO BLOCK
                      TEMPNAM=SHIFT(PTRNAM,12)
                      IF(TEMPNAM.NE.PLIMBO) GO TO 5
                C-PTRIN IS A LIMBO BLOCK.  BEFORE RETURNING DELETE EQCUR AS AN EQ/PTR
                C-OF PTRIN.  FIRST, FETCH ALL EQ/PTRS OF PTRIN
45              C-DELETE EQCUR FROM EQ/PTRIN
                      I=0
                 2    CONTINUE
                C2    PTRIN+ELEM/99 (."I=I+1" =EQIN/2,-.I)
                      LVVAL=PTRIN
50                    LVVARG=LVVAL
                      LVFUNC=ELEM
                      CALL LVFIND
                      IF(LVVTR .EQ. -1) GO TO     99
                      LVV 1=LVVAL
55                    LVV 2=LVFUNC
                      LVV 3=LVVARG
                      I=I+1
                      LVVPOS=   I
                      CALL LVFNV(LV  1)
60                    LVVTR=-1
                      IF(LVVAL .EQ. EQIN
                     +) LVVTR=1
                      IF(LVVTR .EQ. -1) GO TO      2
                      LVFUNC=LVV 2
65                    LVVARG=LVV 3
                      CALL LVFIND
                      LVVPOS=   I
                      CALL LVFNV(LV  2)
                      CALL LVDLTI
70                    RETURN
```

```
C-SET PTCUR AND EQCUR
  5    PTCUR = PTRIN
       EQCUR = EQIN
C-DELETE PTR/POINTER OF EQCUR TO PTCUR
75     C-SET EQLST TO EQCUR
       C-GET NEW EQCUR BY FETCHING PARENT OF CURRENT EQCUR AND PLACING ITS
       C-VALUE IN EQCUR
 10    CONTINUE
       C10    EQCUR*EQLST (-PTR,+PR*EQCUR+LVNAME*HOLNAM)
80           LVVAL=EQCUR
             LVVARG=LVVAL
             EQLST=LVVAL
             LVV 1=LVVARG
             LVFUNC=PTR
85           CALL LVDLET
             LVVAL=LVV 1
             LVVARG=LVVAL
             LVFUNC=PR
             CALL LVFIND
90           EQCUR=LVVAL
             LVVARG=LVVAL
             LVFUNC=LVNAME
             CALL LVFIND
             HOLNAM=LVVAL
95     C-WHAT IS THE BLOCKTYPE OF EQCUR
             ARRAY1=HOLNAM.AND.777700000000000000B
       C-EQCUR IS NOT AN EQ BLOCK TYPE-DONE-RETURN
             IF(ARRAY1.NE.PEQ) RETURN
       C-EQCUR IS AN EQ BLOCK TYPE.  SET PTLST TO PTCUR
100          PTLST = PTCUR
       C-SET A NEW PTCUR BY PLACING IN IT THE VALUE OF PTR OF EQCUR
       C      EQCUR+PTR/99 *PTCUR=PTRIN/30
             LVVAL=EQCUR
             LVVARG=LVVAL
105          LVFUNC=PTR
             CALL LVFIND
             IF(LVVTR .EQ. -1) GO TO     99
             PTCUR=LVVAL
             LVVARG=LVVAL
110          LVVTR=-1
             IF(LVVAL .EQ. PTRIN
            +) LVVTR=1
             IF(LVVTR .EQ. -1) GO TO     30
       C-THE VALUE OF PTCUR IS PTRIN.  GET THE NUMBER OF DOWN POINTERS(EQ*S) OF
115    C-BLOCK EQCUR.
       C-LENGTH IS 1, SO GO BACK ,SET EQCUR,ETC., AND GET NEXT ONE.
       C-LENGTH  GT  1, DONE--RETURN
       C-LENGTH IS  LT  1--ERROR--STOP
       C      EQCUR+EQ/98 .2/10/RETURN
120          LVVAL=EQCUR
             LVVARG=LVVAL
             LVFUNC=EQ
             CALL LVFIND
             IF(LVVTR .EQ. -1) GO TO     98
125          LVVPOS=       2
             CALL LVFNV(LV  3)
             IF(LVVTR .EQ. -1) GO TO     ..
             IF(LVVTR .NE. -1) RETURN
 98    WRITE(ICONW,20)
130    20    FORMAT(1H ,30HERROR-EQ*S OF EQCUR MUST EXIT )
             CALL CONRLDR(4LFLFN,IERR,LFN)
             IF(IERR.NE.0) CALL ERR(4HFLFN,IERR)
             STOP
```

```
C-THE VALUE OF PTRIN IS NOTONE.  SET IDO TO ZERO
  30  IDO = 0
C-IS PTLST EQUAL TO NOTONE
      IF(PTLST.EQ.PNOT1) GO TO 80
C-PTLST IS NOT NOTONE.  DELETE EQLST FROM EQ PTRS OF PTLST.  FIRST MUST
C-FETCH ALL EQCS FROM PTLST
C-NOW TRY TO DELETE EQLST
      I=0
 70   CONTINUE
C70   PTLST+ELEM/40 (."I=I+1"=EQLST/70,-.I)
      LVVAL=PTLST
      LVVARG=LVVAL
      LVFUNC=ELEM
      CALL LVFIND
      IF(LVVTR .EQ. -1) GO TO     40
      LVV 1=LVVAL
      LVV 2=LVFUNC
      LVV 3=LVVARG
      I=I+1
      LVVPOS=    I
      CALL LVFNV(LV  4)
      LVVTR=-1
      IF(LVVAL .EQ. EQLST
     +) LVVTR=1
      IF(LVVTR .EQ. -1) GO TO     70
      LVFUNC=LVV 2
      LVVARG=LVV 3
      CALL LVFIND
      LVVPOS=    I
      CALL LVFNV(LV  5)
      CALL LVDLTI
      GO TO 80
C-PTLAT DOES NOT HAVE ANY EQ POINTERS OR EQLST IS NOT AMONG THOSE THAT
C-ARE THERE.  ERROR--ABORT
  40  WRITE(ICONW,50)
  50  FORMAT(1H ,35HEQLST WAS NOT THERE TO DELETE-ABORT  )
      CALL COMRLDR(4LFLFN,IERR,LFN)
      IF(IERR.NE.0) CALL ERR(4HFLFN,IERR)
      STOP
C-EQLST NOT AMONG EQ'S OF EQCUR.  NEED TO KNOW HOW MANY THERE ARE(LEN)
 80   CONTINUE
C80   EQCUR+EQ/99 .2/85/120
      LVVAL=EQCUR
      LVVARG=LVVAL
      LVFUNC=EQ
      CALL LVFIND
      IF(LVVTR .EQ. -1) GO TO     99
      LVVPOS=        2
      CALL LVFNV(LV  6)
      IF(LVVTR .EQ. -1) GO TO     85
      IF(LVVTR .NE. -1) GO TO    120
```

```
185         C-LEN IS ONE.  MUST DELETE EQLST FROM EQ'S OF PTCUR
            C- IF PTCUR DOES NOT HAVE ANY EQ POINTERS OR EQLST IS NOT AMONG THOSE
            C-EQ'S THAT DO EXIST.  ERROR-ABORT
             85    I=0
             110   CONTINUE
190         C110  PTCUR+ELEM/40(."I=I+1" /40 =EQLST/110,-.I)
                  LVVAL=PTCUR
                  LVVARG=LVVAL
                  LVFUNC=ELEM
                  CALL LVFIND
195               IF(LVVTR .EQ. -1) GO TO     40
                  LVV 1=LVVAL
                  LVV 2=LVFUNC
                  LVV 3=LVVARG
                  I=I+1
200               LVVPOS=   I
                  CALL LVFNV(LV  7)
                  IF(LVVTR .EQ. -1) GO TO     40
                  LVVARG=LVVAL
                  LVVTR=-1
205               IF(LVVAL .EQ. EQLST
                 +) LVVTR=1
                  IF(LVVTR .EQ. -1) GO TO    110
                  LVFUNC=LVV 2
                  LVVARG=LVV 3
210               CALL LVFIND
                  LVVPOS=   I
                  CALL LVFNV(LV  8)
                  CALL LVOLTI
                  GO TO 170
215         C-LEN IS GREATER THAN ONE  NOW HAVE LOOP TO SEE ABOUT THE PTR OF EACH
            C-SUBEQ OF EQCUR
            C-SEE IF THE ITH SUBEQ IS EQLST.  IF IT IS CONTINUE
             120   I=0
             121   CONTINUE
220         C121  EQCUR+EQ/99 ."I=I+1"/140 =EQLST'BN//121
                  LVVAL=EQCUR
                  LVVARG=LVVAL
                  LVFUNC=EQ
                  CALL LVFIND
225               IF(LVVTR .EQ. -1) GO TO     99
                  I=I+1
                  LVVPOS=   I
                  CALL LVFNV(LV  9)
                  IF(LVVTR .EQ. -1) GO TO    140
230               LVVARG=LVVAL
                  LVVTR=-1
                  IF(LVVAL .EQ. EQLST
                 +) LVVTR=1
                  BN=LVVAL
235               IF(LVVTR .NE. -1) GO TO    121
            C-CHECK ON IDO
                  IF(IDO.NE.0) GO TO 130
            C-IDO IS ZERO.  SET IT TO ONE AND SET SAME TO PTR OF DATA(I)
                  IDO = 1
240         C     BN+PTR'SAME
                  LVVAL=BN
                  LVVARG=LVVAL
                  LVFUNC=PTR
                  CALL LVFIND
245               SAME=LVVAL
                  GO TO 140
```

```
      C-IDO WAS NOT ZERO.  SET PTRSUB TO PTR OF DATA(I)
      C-IF PTRSUB IS NOT EQUAL TO SAME, DONE--RETURN
       130  CONTINUE
250   C130  BN+PTR/99'PTRSUB=SAME/RETURN/121
            LVVAL=BN
            LVVARG=LVVAL
            LVFUNC=PTR
            CALL LVFIND
255         IF(LVVTR .EQ. -1) GO TO     99
            PTRSUB=LVVAL
            LVVARG=LVVAL
            LVVTR=-1
            IF(LVVAL .EQ. SAME
260        +) LVVTR=1
            IF(LVVTR .EQ. -1) RETURN
            IF(LVVTR .NE. -1) GO TO    121
      C-IF SAME IS NOTONE, DONE--RETURN
       140  IF(SAME.EQ.PHOT1) RETURN
265   C-SAME IS NOT NOTONE.  MUST DELETE EQ OF SAME POINTERS FROM ALL EQCS
      C-UNDER PARENT(EQCUR).  FIRST, HAVE ALL EQCS OF EQCUR FROM ABOVE IN
      C-ARRAY DATA.  NEXT, MUST FETCH ALL EQ PTRS OF SAME.
            I=0
       150  CONTINUE
270   C150  EQCUR+EQ/99 ."I=I+1"/160 'DATA
            LVVAL=EQCUR
            LVVARG=LVVAL
            LVFUNC=EQ
            CALL LVFIND
275         IF(LVVTR .EQ. -1) GO TO     99
            I=I+1
            LVVPOS=   I
            CALL LVFNV(LV 10)
            IF(LVVTR .EQ. -1) GO TO    160
280         DATA=LVVAL
      C-NOW GO THROUGH PROCESS OF DELETING FROM SAME ALL EQ PTRS WHICH ARE
      C-SUBEQ'S OF EQCUR
        160 J=0
       155  CONTINUE
285   C155  SAME+ELEM/170(,"J=J+1"/150=DATA/150,-,J)
            LVVAL=SAME
            LVVARG=LVVAL
            LVFUNC=ELEM
            CALL LVFIND
290         IF(LVVTR .EQ. -1) GO TO    170
            LVV 1=LVVAL
            LVV 2=LVFUNC
            LVV 3=LVVARG
            J=J+1
295         LVVPOS=   J
            CALL LVFNV(LV 11)
            IF(LVVTR .EQ. -1) GO TO    150
            LVVARG=LVVAL
            LVVTR=-1
300         IF(LVVAL .EQ. DATA
           +) LVVTR=1
            IF(LVVTR .EQ. -1) GO TO    150
            LVFUNC=LVV 2
            LVVARG=LVV 3
305         CALL LVFIND
            LVVPOS=   J
            CALL LVFNV(LV 12)
            CALL LVDLTI
            GO TO 150
310   C-ADD EQCUR AS AN EQ POINTER OF SAME
       170  IF(PTR.NE.SW) GO TO 175
            ARRAY=ITEMPTR
            GO TO 176
```

```
      175 ARRAY=EQ
315    178  CONTINUE
     C178  SAME ARRAY EQCUR
           LVVAL=SAME
           LVVARG=LVVAL
           LVFUNC=ARRAY
320        CALL LVFIND
           LVVALS(1)=EQCUR
           CALL LVNSRT
     C-SET PTR OF EQCUR TO SAME
     C     EQCUR PTR -.1 SAME
325        LVVAL=EQCUR
           LVVARG=LVVAL
           LVFUNC=PTR
           CALL LVFIND
           CALL LVFNV(LV 13)
330        LVVALS(1)=SAME
           CALL LVDSIM
     C-SET EQLST TO EQCUR
           EQLST = EQCUR
     C-SET VALUE OF EQCUR TO THE PARENT NAME OF THE PRESENT EQCUR
335  C     EQCUR+PR/99'EQCUR+LVNAME'HOLNAM
           LVVAL=EQCUR
           LVVARG=LVVAL
           LVFUNC=PR
           CALL LVFIND
340        IF(LVVTR .EQ. -1) GO TO     99
           EQCUR=LVVAL
           LVVARG=LVVAL
           LVFUNC=LVNAME
           CALL LVFIND
345        HOLNAM=LVVAL
     C-EQCUR IS NOT AN EQ BLOCK-DONE--RETURN
           ARRAY1=HOLNAM.AND.777700000000000000000B
            IF(ARRAY1.NE.PEQ) RETURN
     C-EQCUR IS AN EQ BLOCK.  SET PTCUR TO PTR OF EQCUR
350  C     EQCUR+PTR/99'PTCUR
           LVVAL=EQCUR
           LVVARG=LVVAL
           LVFUNC=PTR
           CALL LVFIND
355        IF(LVVTR .EQ. -1) GO TO     99
           PTCUR=LVVAL
     C-GO TO WHERE GETTING THE SUBEQ'S OF EQCUR AND RESUME PROCESS FROM
     C-THERE.
           GO TO 80
360  C    DESIRED POINTER NOT FOUND - ERROR
       99  CALL ERR(5HFETCH,IERR)
           RETURN
     25001 CONTINUE
           LV 1=0
365        LV 2=0
           LV 3=0
           LV 4=0
           LV 5=0
           LV 6=0
370        LV 7=0
           LV 8=0
           LV 9=0
           LV10=0
           LV11=0
375        LV12=0
           LV13=0
           GO TO 25000
           END

STATISTICS
  PROGRAM LENGTH                10018     513
  CM LABELED COMMON LENGTH        468      38
```

REFERENCES

1. Gorham, W. and T. Rhodes, "COMRADE - The Computer-Aided Design Environment Project, An Introduction," DTNSRDC Report 76-0001 (Nov 1976).

2. Thomson, B., "Plex Data Structure for Integrated Ship Design," Presented at the 1973 National Computer Conference, New York, (Jun 1973). American Federation of Information Processing Societies proceedings, pp. 347-352.

3. Bandurski, A. and D. Jefferson, "Enhancements to the Relational Model for Computer-Aided Ship Design," DTNSRDC Report 4759 (Oct 1975).

4. Bandurski, A. and D. Jefferson, "Data Description for Computer-Aided Design," DTNSRDC Report 4750 (Sep 1975).

5. Berkowitz, S., "Design Trade-Offs for a Software Associative Memory," NSRDC Report 3531 (May 1973).

6. Zaritsky, I., "GIRS - (Graph Information Retrieval System) Users Manual," (to be published).

7. Bandurski, A. and M. Wallace, "COMRADE Data Management System - Storage and Retrieval Techniques," Presented at the 1973 National Computer Conference, New York, (Jun 1973), American Federation of Information Processing Societies proceedings, pp. 353-357.

8. Willner, S., et al, "COMRADE Data Management System," Presented at the 1973 National Computer Conference, New York (Jun 1973), American Federation of Information Processing Societies proceedings, pp. 339-345.

9. Martin, J., "Computer Data-Base Organization," Prentice-Hall, Inc., New Jersey (1975), pp. 100-1.

10. "CODASYL Data Description Language," U.S. Department of Commerce, NBS Handbook 113 (Jun 1973), pp. 2.11-2.12.

11. Berkowitz, S., "Graph Information Retrieval Language; Programming Manual for FORTRAN Complement," NSRDC Report 4137 (Jun 1973).

12. Ash, W. and E. Sibley, "TRAMP, An Interpretive Associative Processor with Deductive Capabilities," Proceedings 23rd National Conference of the ACM, pp. 143-156 (1968).

13. Hewitt, C., "Description and Theoretical Analysis (using schemata) of PLANNER," MIT Artificial Intelligence Laboratory AI-TR-258 (1972).

14. McDermott, D.V. and G.J. Sussman, "The CONNIVER Reference Manual," MIT Artificial Intelligence Laboratory AIM-259a (1974).

15. Martin, R. and C. Bell, "A Primer for the COMRADE Data Management System," NSRDC Report 4605 (Jan 1975).

16. Rhodes, T., "The Computer-Aided Design Environment Project (COMRADE)," Presented at the 1973 National Computer Conference, New York (Jun 1973). American Federation of Information Processing Societies proceedings, pp. 319-324.

INITIAL DISTRIBUTION

Copies	
1	CHONR/430D, M. Denicoff
1	NRL
1	NSWC
1	NUSC
1	NOSC
1	NAVSUP/0414, G. Bernstein
3	NAVSEC 1 SEC 6114 1 SEC 6178D
1	Rome Air Development Center
12	DDC

CENTER DISTRIBUTION

Copies	Code	Name
1	18	G. Gleissner
1	1802.2	F. Frenkiel
1	1803	S. Rainey
1	1804	L. Avrunin
1	1805	E. Cuthill
1	1806	J. Pulos
2	1809	D. Harris
1	182	A. Camara
1	1821	D. Jefferson
1	1822	T. Rhodes
1	1824	S. Berkowitz
15	1824	I. Zaritsky
1	1826	L. Culpepper
1	1828	C. Godfrey
1	1828	W. Gorham, Jr.
1	1828	M. Wallace
1	1828.1	I. Datz
1	184	H. Lugt
1	184.1	H. Feingold
1	1843	J. Schot
1	1844	S. Dhir
1	1844	J. McKee
1	185	T. Corin
1	1851	J. Brainin
1	1853	B. Thompson
1	1854	H. Sheridan
1	1855	R. Brengs
1	1856	M. Skall
1	187	M. Zubkoff
1	187	R. Ploe
1	189	G. Gray
1	189.1	N. Taylor
10	5214.1	Reports Distribution
1	522.1	
1	522.2